

Secure Fingerprint Alignment and Matching Protocols

Fattaneh Bayatbabolghani
Computer Science and Engineering
University of Notre Dame
fbayatba@nd.edu

Marina Blanton
Computer Science and Engineering
University at Buffalo (SUNY)
mblanton@buffalo.edu

Mehrdad Aliasgari
Computer Engineering and Computer Science
California State University, Long Beach
mehrdad.aliasgari@csulb.edu

Michael Goodrich
Computer Science
University of California, Irvine
goodrich@uci.edu

Abstract

We present three private fingerprint alignment and matching protocols, based on what are considered to be the most precise and efficient fingerprint recognition algorithms, which use *minutia* points. Our protocols allow two or more honest-but-curious parties to compare their respective privately-held fingerprints in a secure way such that they each learn nothing more than an accurate score of how well the fingerprints match. To the best of our knowledge, this is the first time fingerprint alignment based on minutiae is considered in a secure computation framework. We build secure fingerprint alignment and matching protocols in both the two-party setting using garbled circuit evaluation and in the multi-party setting using secret sharing techniques. In addition to providing precise and efficient secure fingerprint alignment and matching, our contributions include the design of a number of secure sub-protocols for complex operations such as sine, cosine, arctangent, square root, and selection, which are likely to be of independent interest.

1 Introduction

Computing securely with biometric data is challenging because biometric identification applications often require accurate metrics and recognition algorithms, but the data involved is so sensitive that if it is ever revealed or stolen the victim may be vulnerable to impersonation attacks for the rest of their life. This risk is particularly true for fingerprints, which have been used since the 19th century for identification purposes and are being used for such purposes ubiquitously today, including for cellphones, laptops, digital storage devices, safes, immigration, and physical building/room access, as well as the classic application of identifying criminals. Thus, there is a strong motivation for fast and secure fingerprint recognition protocols that protect fingerprint data but nevertheless allow for highly accurate scores for fingerprint comparison.

The setting we consider in this paper is one where the parties involved are honest-but-curious (although, in some cases, malicious parties can also be tolerated using known hardening techniques). That is, two or more parties hold private fingerprint data that they would like to compare in a fast way that provides an accurate score for the comparison but does not reveal the actual fingerprint data. For example, the computation could involve comparing multiple pairs of fingerprint representations stored in two privacy-sensitive databases (e.g., one owned by the FBI and the other owned by a university), or it could involve a single comparison of a suspected criminal's fingerprint to a fingerprint found at a crime scene.

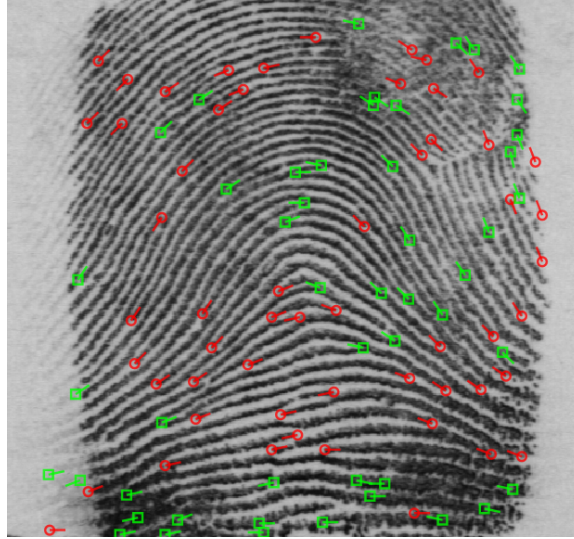


Figure 1: A set of minutiae with orientations. The image is generated by NIST’s Fingerprint Minutiae Viewer (FpMV) software [1] using a fingerprint from NIST’s Special Database 4 [2].

According to accepted best practices (e.g., see [42, 44, 50]) the most accurate *fingerprint recognition* algorithms are based on the use of *minutia* points, which are fingerprint “landmarks,” such as where two ridges merge or split. (See Figure 1.) Such *geometric* recognition algorithms generally involve two main steps: *alignment* and *matching*. Alignment involves computing a geometric transformation to best “line up” two sets of minutiae and matching involves scoring the similarity of two sets of (hopefully overlapping) minutia points. Alignment is necessary for the meaningful and accurate application of the matching step, yet, to the best of our knowledge, all previous publications on secure fingerprint recognition, except for a paper by Kerschbaum *et al.* [36], focus exclusively on the matching step, possibly because of the computational difficulty of finding a good alignment. Instead, our approach is on the design of efficient protocols for the entire fingerprint recognition process, including both the alignment and matching steps, so as to provide secure protocols for accurate fingerprint recognition algorithms used in practice.

In this work, we focus on three algorithms that compare fingerprints using both alignment and matching:

1. A simple geometric transformation algorithm that searches for the maximum matching by considering each pair of minutiae [42].
2. An algorithm aligns fingerprints based on high curvature points [44].
3. An algorithm based on a spectral minutia representation [50].

These algorithms were selected due to their precision, speed, and/or popularity, with the goal of building efficient and practical security solutions. Our main contributions can be summarized as follows:

- We design new secure sine, cosine, and arctangent sub-protocols for fixed-point arithmetic, for both two-party and multi-party cases.

- We design a new secure square-root sub-protocol for fixed-point arithmetic. Our solution is for the two-party setting and is based on Goldschmidt’s method with the last iteration replaced by Newton-Raphson’s method to eliminate accumulated errors.
- We design a new secure sub-protocol for selecting the f th smallest element in a set of comparable elements. Our method is based on an efficient square-root sampling algorithm.
- We build three secure and efficient protocols for fingerprint alignment and matching, based on the use of the above new building blocks applied to the three well-known fingerprint recognition algorithms mentioned above. The constructions work in both two-party and multi-party settings.
- We implement one of the secure fingerprint recognition protocols and show its efficiency in practice.

Our constructions are presented in the semi-honest model. However, a number of available techniques can be used to achieve security in the malicious model (e.g., [28, 23, 4] and others in the multi-party secret sharing setting and [39] among others in the two-party garbled circuits setting).

In what follows, we describe related work in Section 2. Section 3 provides necessary fingerprint background, including the three fingerprint comparison algorithms which are the focus of this work. Section 4 describes the problem statement, security model, and frameworks on which we rely to realize secure fingerprint comparisons. Then Section 5 gives an overview of existing secure building blocks used in this work and describes new secure sub-protocols not available in the prior literature that we design to enable secure fingerprint comparisons. Our main protocols for the three selected fingerprint alignment and matching algorithms are given in Section 6. Experimental evaluation of our secure realization of the third spectral minutia representation algorithm in two different secure computation frameworks is given in Section 7. Lastly, Section 8 concludes this work.

2 Related Work

Work on secure two-party and multi-party computations is extensive and such protocols have been shown to be effective for privacy-preserving evaluation of various functions (e.g., see [25, 18, 41, 24]). Following the seminal work of Yao [51], it is generally known today that any computable function can be securely evaluated, albeit with varying degrees of practicality. For instance, there are a variety of general approaches that typically represent the function to be evaluated as a Boolean or arithmetic circuit (e.g., see [18, 41, 24]). To date, such general approaches have typically not resulted in protocols that are efficient in practice. Thus, there are also a wide variety of custom function-specific protocols that target the design of more efficient secure solutions than what the generic approaches provide, including protocols for biometric identification for several different modalities, such as iris [15] and face [26] recognition.

The first work to treat secure fingerprint comparisons in particular is due to Barni *et al.* [7]. Their method utilizes the FingerCode representation of fingerprints (which uses texture information from a fingerprint) and they secure their method using a homomorphic encryption scheme. FingerCode-based fingerprint comparisons can be implemented efficiently; hence, the method of Barni *et al.* is relatively fast. Unfortunately, the FingerCode approach is not as discriminative as other fingerprint recognition algorithms (in particular, minutia-based algorithms) and is not considered suitable for fingerprint-based identification suitable for criminal trials.

Huang *et al.* [33] propose a privacy-preserving protocol for biometric identification that is focused on fingerprint matching and uses homomorphic encryption and garbled circuit (GC) evaluation. In their fingerprint matching protocol, the FingerCode representation is utilized and the matching score is computed using the Euclidean distance. They provide some optimizations, such as using off-line execution and fewer circuit gates, which make their solution more efficient than prior work. Nevertheless, their method still suffers from the lack of accuracy derived from being based on the FingerCode representation.

Blanton and Gasti [15, 16] also improve the performance of secure fingerprint comparisons based on FingerCodes and additionally provide the first privacy-preserving solution for minutia-based fingerprint matching. Their methods use a combination of homomorphic encryption and GC evaluation, but assume that fingerprints are independently pre-aligned. That is, they treat the matching step only, not the more difficult alignment step. To compare two fingerprints, T and S , consisting of pre-aligned sets of m and n minutia points, respectively, their algorithm considers each point t_i of T in turn, determines the list of points in S within a certain distance and orientation from s_i that have not yet been paired up with another point in T . If this list is not empty, t_i is paired up with the closest point on its list. The total number of paired up points is the size of matching, which can consequently be compared to a threshold to determine whether the fingerprints are related or not. Although their method is lacking in the alignment step, we use similar logic for computing matching between two transformed fingerprints in our protocols.

Shahandashti *et al.* [47] also propose a privacy-preserving protocol for minutia-based fingerprint matching, which uses homomorphic encryption and is based on evaluation of polynomials in encrypted form. The complexity of their protocol is substantially higher than that of Blanton and Gasti [16], however, and their method can also introduce an error when a minutia point from one fingerprint has more than one minutia point from the other fingerprint within a close distance and orientation from it.

More recently, Blanton and Saraph [17] introduce a privacy-preserving solution for minutia-based fingerprint matching that formulates the problem as determining the size of the maximum flow in a bipartite graph, which raises questions of practicality for this method. The algorithm is guaranteed to pair the minutiae from S with the minutiae from T in such a way that the size of the pairing is maximal (which previous solutions could not achieve). The algorithm can be used in both two-party and multi-party settings, but only the two-party protocol based on GC evaluation was implemented.

Lastly, Kerschbaum *et al.* [36] propose a private fingerprint verification protocol between two parties that includes both alignment and matching steps. Unfortunately, the solution leaks information about fingerprint images used and the authors also used a simplified alignment and matching computation that is not as robust to fingerprint variations as other algorithms.

3 Fingerprint Background

A fingerprint represents an exterior surface of a finger. All features extracted from a fingerprint image are intended to allow one to uniquely identify the individual who owns the fingerprint. One of the features most commonly used for fingerprint recognition is *minutiae*, which are represented as a set of points in the two-dimensional plane (possibly with an added angle orientation). Minutiae points generally correspond to fingerprint ridge endings or branches, and are typically represented by the following elements [42]: 1) an x -coordinate, 2) a y -coordinate, 3) an optional orientation, θ , measured in degrees, and 4) an optional minutia type. Some algorithms might store additional information in the fingerprint representation, e.g., the relationship of each minutia point to a

Algorithm 1: Fingerprint recognition based on geometrical transformation

Input: Two fingerprints $T = \{t_i = (x_i, y_i, \theta_i)\}_{i=1}^m$ and $S = \{s_i = (x'_i, y'_i, \theta'_i)\}_{i=1}^n$.

Output: The largest number of matching minutiae, C_{max} , and the corresponding alignment.

1. Initialize $C_{max} = 0$.
 2. For $i = 1, \dots, m$ and $j = 1, \dots, n$, use (t_i, s_j) as a reference pair and perform:
 - (a) Compute transformation $\Delta x = x'_j - x_i$, $\Delta y = y'_j - y_i$, and $\Delta \theta = \theta'_j - \theta_i$.
 - (b) Transfer each minutia $s_k \in S$ as $x''_k = \cos(\Delta \theta) \cdot x'_k + \sin(\Delta \theta) \cdot y'_k - \Delta x$,
 $y''_k = -\sin(\Delta \theta) \cdot x'_k + \cos(\Delta \theta) \cdot y'_k - \Delta y$, and $\theta''_k = \theta'_k - \Delta \theta$ and save the result as
 $S'' = \{s''_i = (x''_i, y''_i, \theta''_i)\}_{i=1}^n$.
 - (c) Compute the number C of matched minutiae between T and S'' (e.g., using Algorithm 2).
 - (d) If $C_{max} < C$, then set $C_{max} = C$ and $(\Delta x_{max}, \Delta y_{max}, \Delta \theta_{max}) = (\Delta x, \Delta y, \Delta \theta)$.
 3. Return C_{max} and its corresponding alignment $(\Delta x_{max}, \Delta y_{max}, \Delta \theta_{max})$.
-

reference point such as the core point representing the center of a fingerprint.

All of the algorithms that we describe are based on minutia points. One of them, however, uses an alternative representation in the form of a spectrum of a fixed size. Thus, the minutiae might be not represented as points, but instead use a spectral representation as detailed later in this section.

Furthermore, one of the fingerprint alignment and matching algorithms upon which we build—namely, the approach that uses curvature information for alignment—relies on an additional type of fingerprint features. In particular, it makes use of high curvature points extracted from a fingerprint image. In addition to storing a number of minutia points, each fingerprint contains a set of high curvature points, each of which is represented as (x, y, w) . Here, first two elements indicate the location of the point and the last element is its curvature value in the range 0–2 (see [44] for the details of its computation).

In the remainder of this section, we describe the three selected fingerprint recognition algorithms without security considerations. All of them take two fingerprints T and S as their input, align the fingerprints, and output the number of matching minutiae between the aligned fingerprint representations. In all algorithms, a fingerprint representation contains a set of minutiae, (t_1, \dots, t_m) for T and (s_1, \dots, s_n) , for S , or an alternative representation derived from the minutiae. Additionally, the second algorithm takes a number of high curvature points stored in a fingerprint as well, and we denote them by $(\hat{t}_1, \dots, \hat{t}_{\hat{m}})$ for T and $(\hat{s}_1, \dots, \hat{s}_{\hat{n}})$ for S .

3.1 Fingerprint Recognition using Brute Force Geometrical Transformation [42]

The first algorithm searches for the best alignment between T and S by considering that minutia, $t_i \in T$, corresponds to minutia, $s_j \in S$, for all possible pairs (t_i, s_j) (called a reference pair) [42]. Once a new reference pair is chosen, the algorithm transforms the minutiae from S using a geometrical transformation and also rotates them, after which it counts the number of matched minutiae. After trying all possible reference pairs, the algorithm chooses an alignment that maximizes the number of matched minutiae and thus increases the likelihood of the two fingerprints to be matched. Algorithm 1 lists the details of this approach.

To implement the matching step, we use an algorithm that iterates through all points in $t_i \in T$ and pairs t_i with a minutia $s_j \in S$ (if any) within a close spatial and directional distance from t_i . When multiple points satisfy the matching criteria, the closest to t_i is chosen, as described in [42, 16]. This computation of the matching step is given in Algorithm 2.

The time complexity of Algorithm 2 is $O(nm)$, and the time complexity of Algorithm 1 is $O(n^2m^2)$ when Algorithm 2 is used as its subroutine.

Algorithm 2: Fingerprint matching

Input: Two fingerprints $T = \{t_i = (x_i, y_i, \theta_i)\}_{i=1}^m$, $S = \{s_i = (x'_i, y'_i, \theta'_i)\}_{i=1}^n$ and thresholds λ and λ_θ for distance and orientation.

Output: The number C of matched minutia pairs between T and S .

1. Set $C = 0$.
 2. Mark each $s_j \in S$ as available.
 3. For $i = 1, \dots, m$, do:
 - (a) Create a list L_i consisting of all available points $s_j \in S$ that satisfy $\min(|\theta_i - \theta'_j|, 360 - |\theta_i - \theta'_j|) < \lambda_\theta$ and $\sqrt{(x_i - x'_j)^2 + (y_i - y'_j)^2} < \lambda$.
 - (b) If L_i is not empty, select the closest minutia s_k to t_i from L_i , mark s_k as unavailable and set $C = C + 1$.
 4. return C .
-

3.2 Fingerprint Recognition using High Curvature Points for Alignment [44]

The second fingerprint recognition algorithm [44] uses high curvature information extracted from the fingerprints to align them. The alignment is based on the iterative closest point (ICP) algorithm [11] that estimates the rigid transformation F between T and S . Once this transformation is determined, it is applied to the minutia points of S . Then the algorithm computes the number of matched minutiae between T and transformed S . The ICP algorithm assumes that the fingerprints are roughly pre-aligned, which can be done, e.g., by aligning each fingerprint independently using its core point, and iteratively finds point correspondences and the transformation between them. To eliminate alignment errors when the overlap between the two sets is partial, [44] suggests using the trimmed ICP (TICP) algorithm [22]. The TICP algorithm ignores a proportion of the points in T whose distances to the corresponding points in S are large, which makes it robust to outliers in the data.

The details of this fingerprint recognition approach are given in Algorithm 3. Steps 1–10 correspond to the TICP algorithm that proceeds with iterations, aligning the fingerprints closer to each in each successive iteration. The algorithm termination parameters λ and γ correspond to the threshold for the total distance between the matched points of T and S and the limit on the number of iterations, respectively. For robustness, the alignment is performed using only a subset of the points (f closest pairs computed in step 4). The optimal motion for transforming one set of the points into the other (treated as two 3-D shapes) is computed using the union quaternion method due to Horn [32], and is given in Algorithm 4. The transformation is represented as a 3×3 matrix R and a vector v of size 3, which are consequently used to align the points in S (step 8).

Once the fingerprints are sufficiently aligned using the high curvature points, the overall transformation between the original and aligned S is computed (step 11). The resulting transformation is applied to the minutia points of S . The new x and y coordinates of the minutiae can be computed directly using the transformation (R, v) , while orientation θ requires a special handling (since it is not used in computing the optimal motion). We compute the difference in the orientation between the original and aligned S as the angle between the slopes of two lines drawn using two points from the original and transformed S , respectively (step 12).

The original ICP algorithm [11] lists expected and worst case complexities of computing the closest points (step 3 of Algorithm 3) to be $O(\hat{m} \log \hat{n})$ and $O(\hat{m}\hat{n})$, respectively. The approach, however, becomes prone to errors in the presence of outliers, and thus in the TICP solution [22] a bounding box is additionally used to eliminate errors. In step 4, we can use f th smallest element selection to find the smallest distances, which runs in linear time (the TICP paper suggests sorting, but sorting results in higher asymptotic complexities). The complexity of Algorithm 4 is $O(k)$ when

Algorithm 3: Fingerprint recognition based on high curvature points for alignment

Input: Two fingerprints consisting of minutiae and high curvature points

$T = (\{t_i = (x_i, y_i, \theta_i)\}_{i=1}^m, \{\hat{t}_i = (\hat{x}_i, \hat{y}_i, \hat{w}_i)\}_{i=1}^{\hat{m}})$ and $S = (\{s_i = (x'_i, y'_i, \theta'_i)\}_{i=1}^n, \{\hat{s}_i = (\hat{x}'_i, \hat{y}'_i, \hat{w}'_i)\}_{i=1}^{\hat{n}})$; a minimum number of matched high curvature points f ; and algorithm termination parameters λ and γ .

Output: The largest number of matching minutiae, C , and the corresponding alignment.

1. Set $S_{LTS} = 0$.
 2. Store a copy of \hat{s}_i 's as $\bar{s}_i = (\bar{x}_i, \bar{y}_i, \bar{w}_i)$ for $i = 1, \dots, \hat{n}$.
 3. For $i = 1, \dots, \hat{n}$, find the closest to \hat{s}_i point \hat{t}_j in T and store their distance d_i . Here, the distance between any \hat{s}_i and \hat{t}_j is defined as $\sqrt{(\hat{x}'_i - \hat{x}_j)^2 + (\hat{y}'_i - \hat{y}_j)^2} + \beta|\hat{w}'_i - \hat{w}_j|$.
 4. Find f smallest d_i 's and calculate their sum S'_{LTS} .
 5. If $S'_{LTS} \leq \lambda$ or $\gamma = 0$, proceed with step 11.
 6. Set $S_{LTS} = S'_{LTS}$.
 7. Compute the optimal motion (R, v) for the selected f pairs using Algorithm 4.
 8. For $i = 1, \dots, \hat{n}$, transform point \hat{s}_i according to (R, v) as $\hat{s}_i = R\hat{s}_i + v$.
 9. Set $\gamma = \gamma - 1$.
 10. Repeat from step 3.
 11. Compute the optimal motion (R, v) for \hat{n} pairs (\bar{s}_i, \hat{s}_i) using Algorithm 4. Let $r_{i,j}$ denote R 's cell at row i and column j and v_i denote the i th element of v .
 12. Compute $c_1 = (\bar{y}_2 - \bar{y}_1)/(\bar{x}_2 - \bar{x}_1)$, $c_2 = (\hat{y}'_2 - \hat{y}'_1)(\hat{x}'_2 - \hat{x}'_1)$, and $\Delta\theta = \arctan((c_1 - c_2)/(1 + c_1c_2))$.
 13. For $i = 1, \dots, n$, apply the transformation to minutia s_i by computing $x'_i = r_{1,1}x'_i + r_{1,2}y'_i + v_1$, $y'_i = r_{2,1}x'_i + r_{2,2}y'_i + v_2$, and $\theta'_i = \theta'_i - \Delta\theta$.
 14. Compute the number, C , of matched minutiae between t_i 's and transformed s_i 's (e.g., using Algorithm 2).
 15. Return C and the transformation (R, v) .
-

run on k input pairs. Thus, the overall complexity of Algorithm 3 (including Algorithms 4 and 2) is $O(\gamma\hat{m}\hat{n} + mn)$, where γ is the upper bound on the number of iterations in the algorithm.

3.3 Fingerprint Recognition based on Spectral Minutiae Representation [50]

The last fingerprint recognition algorithm by Xu et al. [50] uses minutia-based representation of fingerprints, but offers greater efficiency than other algorithms because of an alternative form of minutia representation. The spectral minutia representation [49] that it uses is a fixed-length feature vector, in which rotation and scaling can be easily compensated for, resulting in efficient alignment of two fingerprints.

In the original spectral minutia representation [49], a set of minutiae corresponds to a real-valued vector of a fixed length D , which is written in the form of a matrix of dimensions $M \times N$ ($= D$). The vector is normalized to have zero mean and unit energy. Thus, we now represent T and S as matrices of dimensions $M \times N$ and denote their individual elements as $t_{i,j}$'s and $s_{i,j}$'s, respectively. In [49], there are two types of minutia spectra (location-based and orientation-based), each with $M = 128$ and $N = 256$, and a fingerprint represented by their combination consists of 65,536 real numbers.

To compare two fingerprints in the spectral representation, two-dimensional correlation is used as a measure of their similarity. Furthermore, to compensate for fingerprint image rotations, which in this representation become circular shifts in the horizontal direction, the algorithm tries a number of rotations in both directions and computes the highest score among all shifts. This alignment and matching algorithm is presented as Algorithm 5. It is optimized to perform shifts from -15 to 15 units (which correspond to rotations from -10° to $+10^\circ$) and computes only 9 similarity scores instead of all 31 of them. Later in this section, we show how the algorithm can be generalized to

Algorithm 4: Union quaternion method for computing optimal motion

Input: n pairs $\{(t_i = (x_i, y_i, z_i), s_i = (x'_i, y'_i, z'_i))\}_{i=1}^n$.

Output: Optimal motion (R, v) .

1. For $i = 1, \dots, n$, compute unit quaternion $q_i = (q_{(i,1)}, q_{(i,2)}, q_{(i,3)}, q_{(i,4)}) = \left(\sqrt{\frac{1+k_i}{2}}, u_i \sqrt{\frac{1-k_i}{2}}\right)$, where $k_i = \frac{x_i \cdot x'_i + y_i \cdot y'_i + z_i \cdot z'_i}{\sqrt{x_i^2 + y_i^2 + z_i^2} \sqrt{x_i'^2 + y_i'^2 + z_i'^2}}$, $u_i = \left(\frac{y_i \cdot z'_i - z_i \cdot y'_i}{\|t_i \times s_i\|}, \frac{z_i x'_i - x_i z'_i}{\|t_i \times s_i\|}, \frac{x_i y'_i - y_i x'_i}{\|t_i \times s_i\|}\right)$, and $\|t_i \times s_i\| = \sqrt{(y_i \cdot z'_i - z_i \cdot y'_i)^2 + (z_i \cdot x'_i - x_i \cdot z'_i)^2 + (x_i \cdot y'_i - y_i \cdot x'_i)^2}$.
 2. Compute the overall unit quaternions $q = [q_1, q_2, q_3, q_4] = q_1 q_2 \dots q_{n-1} q_n$ by executing multiplication from left to right, where $q_j q_{j+1} = [q_{(j,1)} q_{(j+1,1)} - v_j \cdot v_{j+1}, q_{(j,1)} v_{j+1} + q_{(j+1,1)} v_j + v_j \times v_{j+1}]$, $v_j = (q_{(j,2)}, q_{(j,3)}, q_{(j,4)})$, and $v_{j+1} = (q_{(j+1,2)}, q_{(j+1,3)}, q_{(j+1,4)})$ for $j = 1, \dots, n-1$.
 3. Compute rotation matrix $R = \begin{bmatrix} q_1^2 + q_2^2 - q_3^2 - q_4^2 & 2(q_2 q_3 - q_1 q_4) & 2(q_2 q_4 + q_1 q_3) \\ 2(q_3 q_2 + q_1 q_4) & q_1^2 - q_2^2 + q_3^2 - q_4^2 & 2(q_3 q_4 - q_1 q_2) \\ 2(q_4 q_2 - q_1 q_3) & 2(q_4 q_3 - q_1 q_2) & q_1^2 - q_2^2 - q_3^2 + q_4^2 \end{bmatrix}$.
 4. Compute transformation vector $v = t - Rs$, where $t = (\sum_{i=1}^n t_i)/n$ and $s = (\sum_{i=1}^n s_i)/n$.
 5. Return (R, v) .
-

Algorithm 5: Fingerprint recognition based on spectral minutia representation

Input: Two real-valued matrices $T = \{t_{i,j}\}_{i=1,j=1}^{M,N}$ and $S = \{s_{i,j}\}_{i=1,j=1}^{M,N}$ and parameter $\lambda = 15$ indicating the maximum amount of rotation.

Output: The best matching score C_{max} and the corresponding alignment.

1. Set $C_{max} = 0$.
 2. For $\alpha = -12, -6, 0, 6, 12$, do:
 - (a) Compute the similarity score between T and S horizontally shifted by α positions as $C_\alpha = \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N t_{i,j} \cdot s_{i,(j+\alpha) \bmod N}$.
 - (b) If $C_\alpha > C_{max}$, set $C_{max} = C_\alpha$, $k = \alpha$, and $\alpha_{max} = \alpha$.
 3. For $\alpha = k - 2, k + 2$, do:
 - (a) Compute the similarity score between T and S shifted by α positions as in step 2(a).
 - (b) If $C > C_{max}$, set $C_{max} = C$, $k' = \alpha$, and $\alpha_{max} = \alpha$.
 4. For $\alpha = k' - 1, k' + 1$, do:
 - (a) Compute the similarity score between T and S shifted by α positions as in step 2(a).
 - (b) If $C > C_{max}$, set $C_{max} = C$ and $\alpha_{max} = \alpha$.
 5. Return C_{max} and α_{max} .
-

work with any amount of shift λ resulting in only $O(\log \lambda)$ score computations.

Xu et al. [50] apply feature reduction to the spectral minutia representation to reduce the size of the feature vector and consequently improve the time of fingerprint comparisons without losing precision. That work describes two types of feature reduction: Column Principle Component Analysis (CPCA) and Line Discrete Fourier Transform (LDFT) feature reductions. Then one or both of them can be applied to the feature vector, with the latter option providing the greatest savings.

The first form of feature reduction, CPCA, reduces the minutia spectrum feature in the vertical direction. After the transformation, the signal is concentrated in the top rows and the remaining rows that carry little or no energy are removed. As a result, a minutia spectrum is represented as a matrix of dimension $M' \times N$ for some $M' < M$. Because the rotation operation commutes with this transformation, the score computation in Algorithm 5 remain unchanged after CPCA feature reduction.

The second form of feature reduction, LDFT, reduces the minutia spectrum in the horizontal direction and returns a feature matrix of dimension $M \times N'$ (or $M' \times N'$ if applied after the CPCA feature reduction) for some $N' < N$. The matrix consists of complex numbers as a result of applying

Fourier transform. After the LDFT, the energy is concentrated in the middle columns and other columns are consequently removed. Shifting matrix S in the resulting representation by α positions now translates into setting its cell at location k, j to $e^{-i\frac{2\pi}{N}j\alpha} \cdot s_{k,j}$, where i indicates the imaginary part of a complex number. Note that circular shifts are no longer applied to matrix rows (including when the LDFT is applied after the CPCA feature reduction).

The resulting matrices of complex numbers T and S are then converted to real-valued matrices and processed using Algorithm 5. If we express a cell of T or S at location k, j as $a_{k,j} + ib_{k,j}$ by separating its real and imaginary parts, then the k th row of T and S is now expressed as a real-valued vector

$$\sqrt{\frac{1}{N}}a_{k,1}, \sqrt{\frac{2}{N}}a_{k,2}, \dots, \sqrt{\frac{2}{N}}a_{k,N}, \sqrt{\frac{2}{N}}b_{k,2}, \dots, \sqrt{\frac{2}{N}}b_{k,N'}.$$

Computing the score between T and S then corresponds to computing the dot product of each k th row of T and S and adding the values across all k . When we use Algorithm 5 to compute the best similarity score, we need to adjust the computation in step 2(a) for the new matrix dimensions and also implement rotation as a multiplication instead of a shift. If we expand the quantity $e^{-i\frac{2\pi}{N}j\alpha}$ using the formula $e^{i\varphi} = \cos(\varphi) + i\sin(\varphi)$, the real part of each score between T and S rotated by α positions now becomes:

$$\begin{aligned} C_\alpha = & \frac{1}{MN^2} \sum_{k=1}^{M'} \left(a_{k,1} \left(a'_{k,1} \cos\left(-\frac{2\pi\alpha}{N}\right) - b'_{k,1} \sin\left(-\frac{2\pi\alpha}{N}\right) \right) + \right. \\ & \left. + 2 \sum_{j=2}^{N'} \left(\cos\left(-\frac{2\pi j\alpha}{N}\right) (a_{k,j}a'_{k,j} + b_{k,j}b'_{k,j}) + \sin\left(-\frac{2\pi j\alpha}{N}\right) (a'_{k,j}b_{k,j} - a_{k,j}b'_{k,j}) \right) \right) \end{aligned} \quad (1)$$

where $t_{k,j} = a_{k,j} + ib_{k,j}$ and original $s_{k,j} = a'_{k,j} + ib'_{k,j}$.

Returning to Algorithm 5, we generalize the algorithm to work for any value of λ so that only $O(\log \lambda)$ score computations are necessary. Let $\lambda = c \cdot 3^k$ for some constants $c \geq 2$ and $k \geq 1$. Our algorithm proceeds in $k + 1$ iterations computing c scores in the first iteration and 2 scores in each consecutive iteration, which results in the total of $c + 2k$ score computations. In the initial iteration 0, we compute c scores at indices $\lambda = \lceil 3^k/2 \rceil, \dots, \lceil 3^k(2c - 1)/2 \rceil$, then determine their maximum C_{max} and the index of the maximum score α_{max} . In iteration $i = 1, \dots, k$, the algorithm computes two scores at indices $\alpha_{max} \pm 3^{k-i}$, determines the maximum of the computed scores and C_{max} , and updates the value of α_{max} as needed. Compared to Algorithm 5, this approach covers 54 shifts using 8 score computations instead of 9 score computations for 31 shifts. The best performance is achieved when $\lambda = 2 \cdot 3^k$ for some k . If λ is not of that form, we could use a different value of c , cover a wider range of shifts than required, or decrease the step size of the algorithm slower than by a factor of 3 (as was done in Algorithm 5), which results in redundant coverage.

In, [50], parameters M' and N' are chosen to retain most of the signal's energy after the transformations (e.g., 90%) and may be dependent on the data set. The complexity of Algorithm 5 for two fingerprints after both types of feature reduction is $O(M'N' \log \lambda)$.

4 Security Model

4.1 Problem Statement and Security Definitions

Because of the variety of settings in which fingerprint recognition may be used, we distinguish between different computation setups and the corresponding security settings.

1. There will be circumstances when two entities would like to compare fingerprints that they respectively possess without revealing any information about their data to each other. This can correspond to the cases when both entities own a fingerprint database and would like to find entries common to both of them or when one entity would like to search a database belonging to a different entity for a specific fingerprint. In these settings, the computation takes the form of secure two-party computation, where the participants can be regarded as semi-honest or fully malicious depending on the application and their trust level.
2. There will also be circumstances when one or more data owners are computationally limited and would like to securely offload their work to more powerful servers or cloud providers (this applies even if all fingerprints used in the computation belong to a single entity). Then multi-party settings that allow for input providers to be different from the computational parties apply. In such settings, the computational parties are typically treated as semi-honest, but stronger security models can also be used for added security guarantees.

Security of any multi-party protocol (with two or more participants) can be formally shown according to one of the two standard security definitions. The first, weaker security model assumes that the participants are semi-honest (also known as honest-but-curious or passive), defined as they follow the computation as prescribed, but might attempt to learn additional information about the data from the intermediate results. The second, stronger security model allows dishonest participants to arbitrarily deviate from the prescribed computation. The focus of this work is on the semi-honest adversarial model, although standard techniques for strengthening security in the presence of fully malicious participants can be used as well. Security in the semi-honest setting is defined as follows.

Definition 1 *Let parties P_1, \dots, P_n engage in a protocol Π that computes function $f(\text{in}_1, \dots, \text{in}_n) = (\text{out}_1, \dots, \text{out}_n)$, where in_i and out_i denote the input and output of party P_i , respectively. Let $\text{VIEW}_\Pi(P_i)$ denote the view of participant P_i during the execution of protocol Π . More precisely, P_i 's view is formed by its input and internal random coin tosses r_i , as well as messages m_1, \dots, m_k passed between the parties during protocol execution: $\text{VIEW}_\Pi(P_i) = (\text{in}_i, r_i, m_1, \dots, m_k)$. Let $I = \{P_{i_1}, P_{i_2}, \dots, P_{i_\tau}\}$ denote a subset of the participants for $\tau < n$ and $\text{VIEW}_\Pi(I)$ denote the combined view of participants in I during the execution of protocol Π (i.e., the union of the views of the participants in I). We say that protocol Π is τ -private in the presence of semi-honest adversaries if for each coalition of size at most τ there exists a probabilistic polynomial time simulator S_I such that $S_I(\text{in}_I, f(\text{in}_1, \dots, \text{in}_n)) \equiv \{\text{VIEW}_\Pi(I), \text{out}_I\}$, where $\text{in}_I = \bigcup_{P_i \in I} \{\text{in}_i\}$, $\text{out}_I = \bigcup_{P_i \in I} \{\text{out}_i\}$, and \equiv denotes computational or statistical indistinguishability.*

4.2 Secure Function Evaluation Frameworks

To realize our constructions in a variety of settings, two secure computation frameworks are of interest to us. In the two-party setting, we build on garbled circuit (GC) evaluation techniques, while in the multi-party setting, we employ linear secret sharing (SS) techniques. These frameworks are briefly described below.

Garbled circuit evaluation. The use of GCs allows two parties P_1 and P_2 to securely evaluate a Boolean circuit of their choice. That is, given an arbitrary function $f(x_1, x_2)$ that depends on private inputs x_1 and x_2 of P_1 and P_2 , respectively, the parties first represent it as a Boolean circuit. One party, say P_1 , acts as a circuit generator and creates a garbled representation of the circuit by associating both values of each binary wire with random labels. The other party, say P_2 , acts as a circuit evaluator and evaluates the circuit in its garbled representation without knowing the

meaning of the labels that it handles during the evaluation. The output labels can be mapped to their meaning and revealed to either or both parties.

An important component of GC evaluation is 1-out-of-2 Oblivious Transfer (OT). It allows the circuit evaluator to obtain wire labels corresponding to its inputs. In particular, in OT the sender (i.e., circuit generator in our case) possesses two strings s_0 and s_1 and the receiver (circuit evaluator) has a bit σ . OT allows the receiver to obtain string s_σ and the sender learns nothing. An OT extension allows any number of OTs to be realized with small additional overhead per OT after a constant number of regular more costly OT protocols (the number of which depends on the security parameter). The literature contains many realizations of OT and its extensions, including very recent proposals such as [45, 34, 5] and others.

The fastest currently available approach for GC generation and evaluation we are aware of is by Bellare et al. [10]. It is compatible with earlier optimizations, most notably the “free XOR” gate technique [38] that allows XOR gates to be processed without cryptographic operations or communication, resulting in lower overhead for such gates. A recent half-gates optimization [52] can also be applied to this construction to reduce communication associated with garbled gates.

Secret sharing. SS techniques allow for private values to be split into random shares, which are distributed among a number of parties, and perform computation directly on secret shared values without computationally expensive cryptographic operations. Of a particular interest to us are linear threshold SS schemes. With a (n, τ) -secret sharing scheme, any private value is secret-shared among n parties such that any $\tau + 1$ shares can be used to reconstruct the secret, while τ or fewer parties cannot learn any information about the shared value, i.e., it is perfectly protected in the information-theoretic sense. In a linear SS scheme, a linear combination of secret-shared values can be performed by each party locally, without any interaction, but multiplication of secret-shared values requires communication between all of them.

In this setting, we can distinguish between the input owner who provide input data into the computation (by producing secret shares), computational parties who conduct the computation on secret-shared values, and output recipients who learn the output upon computation termination (by reconstructing it from shares). These groups can be arbitrarily overlapping and be composed of any number of parties as long as there are at least 3 computational parties.

In the rest of this work, we assume that Shamir SS [48] is used with $\tau < n/2$ in the semi-honest setting for any $n \geq 3$.

5 Secure Building Blocks

To be able to build secure protocols for different fingerprint recognition algorithms, we will need to rely on secure algorithms for performing a number of arithmetic operations, which we discuss in this section. Most of the computation described in Section 3 is performed on real numbers, while for some of its components integer arithmetic will suffice (e.g., Algorithm 2 can be executed on integers when its inputs are integer values). Complex numbers are represented as a pair (a real part and an imaginary part) of an appropriate data type.

For the purposes of this work, we choose to use fixed-point representation for real numbers. Operations on fixed-point numbers are generally faster than operations using floating-point representation (see, e.g., [3]), and fixed-point representation provides sufficient precision for this application.

In the two-party setting based on GCs, complexity of an operation is measured in the number of non-free (i.e., non-XOR) Boolean gates. In the multi-party setting based on SS, we count the total number of elementary interactive operations as well as the number of sequential interactive

operations (i.e., rounds). Throughout this work, we use notation $[x]$ to denote that the value of x is protected and not available to any participant in the clear.

In what follows, we start by listing building blocks from prior literature that we utilize in our solutions (Section 5.1) and then proceed with presenting our design for a number of new building blocks for which we did not find secure realizations in the literature (Section 5.2).

5.1 Known Building Blocks

Some of the operations used in the computation are elementary and are well-studied in the security literature (e.g., [20, 21, 13, 14]), while others are more complex, but still have presence in prior work (e.g., [13, 21, 12, 6]). When it is relevant to the discussion, we assume that integers are represented using ℓ bits and fixed-point values are represented using the total of ℓ bits, k of which are stored after the radix point (and thus $\ell - k$ are used for the integer part).

- *Addition* $[c] \leftarrow [a] + [b]$ and *subtraction* $[c] \leftarrow [a] - [b]$ are considered free (non-interactive) using SS using both fixed-point and integer representations [21]. Their cost is ℓ non-free gates for ℓ -bit a and b [37] using GCs for both integer and fixed-point representations.
- *Multiplication* $[c] \leftarrow [a] \cdot [b]$ of integers involves 1 interactive operation (in 1 round) using SS. For fixed-point numbers, truncation of k bits is additionally needed, resulting in $2k + 2$ interactive operations in 4 rounds (which reduces to 2 rounds after pre-computation) [21]. Using GCs, multiplication of ℓ -bit values (both integer and fixed-point) involves $2\ell^2 - \ell$ non-free gates using the traditional algorithm [37]. This can be reduced using the Karatsuba's method [35], which results in fewer gates when $\ell > 19$ [31]. Note that truncation has no cost in Boolean circuits.
- *Comparison* $[c] \leftarrow \text{LT}([a], [b])$ that tests for $a < b$ (and other variants) and outputs a bit involves $4\ell - 2$ interactive operations in 4 rounds (which reduces to 3 rounds after pre-computation) using SS [20] (alternatively, $3\ell - 2$ interactive operations in 6 rounds). This operation costs ℓ non-free gates using GCs [37]. Both implementations work with integer and fixed-point values of length ℓ .
- *Equality testing* $[c] \leftarrow \text{EQ}([a], [b])$ similarly produces a bit and costs $\ell + 4\log \ell$ interactive operations in 4 rounds using SS [20]. GC-based implementation requires ℓ non-free gates [38]. The implementations work with both integer and fixed-point representations.
- *Division* $[c] \leftarrow \text{Div}([a], [b])$ is available in the literature based on different underlying algorithms and we are interested in the fixed-point version. A fixed-point division based on SS is available from [21] which uses Goldschmidt's method. The algorithm proceeds in ξ iterations, where $\xi = \lceil \log_2(\frac{\ell}{3.5}) \rceil$. The same underlying algorithm could be used to implement division using GCs, but we choose to use the readily available solution from [13] that uses the standard (shift and subtraction) division algorithm. The complexities of these implementations are given in Table 1.
- *Integer to fixed-point conversion* $[b] \leftarrow \text{Int2FP}([a])$ converts an integer to the fixed-point representation by appending a number of zeros after the radix point. It involves no interactive operations using SS and no gates using GCs.
- *Fixed-point to integer conversion* $[b] \leftarrow \text{FP2Int}([a])$ truncates all bits after the radix point of its input. It involves no gates using GCs and costs $2k + 1$ interactive operations in 3 rounds to truncate k bits using SS [21].

- *Conditional statements with private conditions* of the form “if $[priv]$ then $[a] = [b]$; else $[a] = [c]$,” are transformed into statements $[a] = ([priv] \wedge [b]) \vee (\neg[priv] \wedge [c])$, where b or c may also be the original value of a (when only a single branch is present). Our optimized implementation of this statement using GCs computes $[a] = ([priv] \wedge ([b] \oplus [c])) \oplus [c]$ with the number of non-XOR gates equal to the bitlength of variables b and c . Using SS, we implement the statement as $[a] = [priv] \cdot ([b] - [c]) + [c]$ using a single interactive operation.
- *Maximum or minimum* of a set $\langle [a_{max}], [i_{max}] \rangle \leftarrow \text{Max}([a_1], \dots, [a_m])$ or $\langle [a_{min}], [i_{min}] \rangle \leftarrow \text{Min}([a_1], \dots, [a_m])$, respectively, is defined to return the maximum/minimum element together with its index in the set. The operation costs $2\ell(m-1) + m + 1$ non-free gates using GCs, where ℓ is the bitlength of the elements a_i . Using SS, the cost is dominated by the comparison operations, giving us $4\ell(m-1)$ interactive operations. Instead of performing comparisons sequentially, they can be organized into a binary tree with $\lceil \log m \rceil$ levels of comparisons. Then in the first iteration, $m/2$ comparisons are performed, $m/4$ comparisons in the second iteration, etc., with a single comparison in the last iteration. This allows the number of rounds to grow logarithmically with m and give us $4\lceil \log m \rceil + 1$ rounds.

When each record of the set contains multiple fields (i.e., values other than those being compared), the cost of the operation increases by $m-1$ non-free gates for each additional *bit* of the record using GCs and by $m-1$ interactive operations for each additional *field element* of the record without increasing the number of rounds.

- *Prefix multiplication* $\langle [b_1], \dots, [b_m] \rangle \leftarrow \text{PreMul}([a_1], \dots, [a_m])$ simultaneously computes $[b_i] = \prod_{j=1}^i [a_j]$ for $i = 1, \dots, m$. We also use an abbreviated notation $\langle [b_1], \dots, [b_m] \rangle \leftarrow \text{PreMul}([a], m)$ when all a_i 's are equal. In the SS setting, this operation saves the number of rounds (with GCs, the number of rounds is not the main concern and multiple multiplications can be used instead of this operation). The most efficient constant-round implementation of *PreMul* for integers is available from [20] that takes only 2 rounds and $3m-1$ interactive operations. The solution, however, is limited to non-zero integers. We are interested in prefix multiplication over fixed-point values and suggest a tree-based solution consisting of fixed-point multiplications, similar to the way minimum/maximum protocols are constructed. This requires $(m-1)(2k+2)$ interactive operations in $2\lceil \log m \rceil + 2$ rounds.
- *Compaction* $\langle [b_1], \dots, [b_m] \rangle \leftarrow \text{Comp}([a_1], \dots, [a_m])$ pushes all non-zero elements of its input to appear before any zero element of the set. We are interested in order-preserving compaction that also preserves the order of the non-zero elements in the input set. A solution from [17] (based on data-oblivious order-preserving compaction in [29]) can work in both GCs and SS settings using any type of input data. In this paper we are interested in the variant of compaction that takes a set of tuples $\langle a'_i, a''_i \rangle$ as its input, where each a'_i is a bit that indicates whether the data item a''_i is zero or not (i.e., comparison of each data item to 0 is not needed). The complexities of this variant are given in Table 1.
- *Array access at a private index* allows to read or write an array element at a private location. In this work we utilize only read accesses and denote the operation as a table lookup $[b] \leftarrow \text{TLookup}(\langle [a_1], \dots, [a_m] \rangle, [ind])$. The array elements a_i might be protected or publically known, but the index is always private. Typical straightforward implementations of this operations include a multiplexer (as in, e.g., [53]) or comparing the index $[ind]$ to all positions of the array and obviously choosing one of them. Both implementations have complexity $O(m \log m)$ and work with GCs and SS techniques and data of different types. Based on our analysis and performance of components of this functionality, a multiplexer-based implementation

Prot.	Secret sharing		Garbled circuits	
	Rounds	Interactive operations	XOR gates	Non-XOR gates
Add/Sub	0	0	4ℓ	ℓ
LT	4	$4\ell - 2$	3ℓ	ℓ
EQ	4	$\ell + 4\log \ell$	ℓ	ℓ
Mul	1 : 4	1 : $2k + 2$	$4\ell^2 - 4\ell$	$2\ell^2 - \ell$
Div	– : $3\log \ell + 2\xi + 12$	– : $1.5\ell \log \ell + 2\ell\xi + 10.5\ell + 4\xi + 6$	$7\ell^2 + 7\ell$	$3\ell^2 + 3\ell$
PreMul	2 : $2\log m + 2$	$3m - 1 : (m - 1)(2k + 2)$	–	–
Max/Min	$4\log m + 1$	$4\ell(m - 1)$	$5\ell(m - 1)$	$2\ell(m - 1)$
Int2FP	0	0	0	0
FP2Int	3	$2k + 1$	0	0
Comp	$\log m + \log \log m + 3$	$m \log m \log \log m + 4m \log m - m + \log m + 2$	$(\ell + 4)m \log m - m\ell - 4\log m + \ell$	$(2\ell + 1)m \log m - 2\ell m + (\ell - 1) \log m + 2\ell$
Sort	$2\log m(\log m + 1) + 1$	$\ell(m - 0.25)(\log^2 m + \log m + 4)$	$1.5m\ell(\log^2 m + \log m + 4)$	$0.5m\ell(\log^2 m + \log m + 4)$
TLookup	5	$m \log m + 4m \log \log m + m$	$m\ell + \log m - \ell$	$m \log m + m(\ell - 1)$

Table 1: Performance of known secure building blocks on integer and fixed-point values.

outperforms the comparison-based implementation for GCs, while the opposite is true for SS-based techniques. We thus report performance of the best option for SS and GC settings in Table 1 .

Each record a_i can be large, in which case the complexity of the operation additionally linearly grows with the size of array elements (or the number of field elements that each array stores in the SS setting).

- *Oblivious sorting* $\langle [b_1], \dots, [b_m] \rangle \leftarrow \text{Sort}([a_1], \dots, [a_m])$ obviously sorts an m -element set. While several algorithms of complexity $O(m \log m)$ are known, in practice the most efficient oblivious sorting is often the Batcher’s merge sort [8]. According to [14], the algorithm involves $\frac{1}{4}m(\log^2 m - \log m + 4)$ compare-and-exchange operations that compare two elements and conditionally swap them.

The complexities of all building blocks are listed in Table 1 , and notation is explained with each respective protocol. All functions with the exception of Int2FP and FP2Int the associated integer and fixed-point variants, performance of which might differ in the SS sharing. Because most protocols exhibit the same performance for integer and fixed-point variants, for the functions with different performance, we list both variants (integer followed by fixed-point) separated by “:”.

5.2 New Building Blocks

To build secure fingerprint recognition algorithms, we also need a number of secure building blocks for rather complex operations that previously have not been sufficiently treated in the literature. Thus, in this section, we present secure constructions for a number of operations, including trigonometric operations, square root, and selection of the f th smallest element of a set.

We explore the available algorithms for computing these functions (e.g., Chebyshev polynomials for trigonometric functions) and build protocols that optimize performance in each of the chosen settings of two-party and multi-party computation. Our optimizations for designing the building blocks as well as the overall protocols focus on the specific costs of the underlying techniques for

secure arithmetic and ensure that we achieve efficiency together with precision and provable security guarantees.

5.2.1 Sine, Cosine, and Arctangent

We now give new secure building blocks for three trigonometric functions, which are sine, cosine, and arctangent. We denote secure protocols as $[b] \leftarrow \text{Sin}([a])$, $[b] \leftarrow \text{Cos}([a])$, and $[b] \leftarrow \text{Arctan}([a])$ for each respective function. The output b is a fixed-point value, while the input a (for sine and cosine) represented in degrees can be either integer or fixed-point. For generality, we will assume that a used with trigonometric functions is also a fixed-point value, while slight optimizations are possible when a is known to be an integer (as in the fingerprint algorithms considered in this work).

There are a variety of different approximation algorithms that can be used for trigonometric functions, many of which take the form of polynomial evaluation. Upon examining the options, we chose to proceed with the polynomials described in [30, Chapter 6], as they achieve good precision using only small degree polynomials. Note that Taylor series of trigonometric functions achieve an approximation when the input is very close to some fixed point. However, in secure computation, we can only assume that an input value belongs to a known range (e.g., $[0, 2\pi]$). Therefore, we choose polynomial approximation over a range for inputs from [30] instead of Taylor series. The polynomials used in [30] for trigonometric functions take the form $P(x^2)$ or $xP(x^2)$ for some polynomial P over variable x (i.e., use only odd or even powers), which requires one to compute only half as many multiplications as in a polynomial with all powers present.

Another very different approach is to precompute the values of these functions for the desired precision and the range of input values and use private table lookup to select the output based on private input. This approach was recently pursued in [46], but has scaling issues for general function evaluation. We elaborate on the possibility of using this approach in Appendix A.1, while this section treats the more general solution based on polynomial evaluation.

The approach used in [30] offers two types of polynomial approximations. The first type uses a regular polynomial $P(x)$ of degree N to approximate the desired function. The second type uses a rational function of the form $P(x)/Q(x)$, where P and Q are polynomials of degree N and M , respectively. For the same desired precision, the second option will yield lower degrees N and M and thus fewer multiplications to approximate the function. This option, however, is undesirable when the division operation is much costlier than the multiplication operation. In our setting, the rational form is preferred in the case of GC evaluation (at least for higher precisions) where multiplication and division have similar costs. However, with SS, the cost of division is much higher than that of multiplication, and we use the first option with regular polynomial evaluation.

It is assumed in [30] that the initial range reduction to $[0, \pi/2]$ is used for the input to trigonometric functions. This means that if input a to sine or cosine is given in degrees, it needs to be reduced to the range $[0, 90]$ and normalized to the algorithm's expectations. Note that it is straightforward to extend the result of the computation to cover the entire range $[0, 2\pi]$ given the output of this function and the original input.

Furthermore, because evaluating trigonometric functions on a smaller range of inputs offers higher precision, it is possible to apply further range reduction and evaluate the function on even a smaller range of inputs, after which the range is expanded using an appropriate formula or segmented evaluation. We refer the reader to [30] for additional detail. For that reason, the tables of polynomial coefficients provided in [30] are for $\sin(\alpha\pi x)$ and $\cos(\beta\pi x)$, where $0 \leq x \leq 1$ and α and β are fixed constants equal to $1/2$ or less. To see the benefit provided by smaller α or β , for example, consider the sine function. Then by using $\alpha = 1/2$ and approximating the function using a regular polynomial of degree 7, we obtain precision to 15.85 decimal places, while with $\alpha = 1/6$

and the same polynomial degree 25.77 decimal places of the output can be computed. However, for simplicity, in this work we suggest to choose α and β equal to $1/2$, as this option will not require non-trivial post-computation to compensate for the effect of computing the function on a different angle.

Based on the desired precision, one can retrieve the minimum necessary polynomial degree used in the approximation to achieve the desired precision, then look up the polynomial coefficients and evaluate the polynomial or polynomials on the input. As mentioned before, for trigonometric functions only polynomials with even or odd powers are used, and we have that sine is approximated as $xP(x^2)$ or $xP(x^2)/Q(x^2)$ (odd powers) and cosine as $P(x^2)$ or $P(x^2)/Q(x^2)$ (even powers). Thus, in when a rational function is used to evaluate sine, we obtain the following secure protocol. It assumes that the input is given in degrees in the range $[0, 360)$.

$[b] \leftarrow \text{Sin}([a])$

1. Compute $[s] = \text{LT}(180, [a])$.
 2. If $([s])$ then $[a] = [a] - 180$.
 3. If $(\text{LT}(90, [a]))$ then $[a] = 180 - [a]$.
 4. Compute $[x] = \frac{1}{90}[a]$ and then $[w] = [x]^2$.
 5. Lookup the minimum polynomial degrees N and M using $\alpha = 1/2$ for which precision of the approximation is at least k bits.
 6. Lookup polynomial coefficients p_0, \dots, p_N and q_0, \dots, q_M for sine approximation.
 7. Compute $([z_1], \dots, [z_{\max(N, M)}]) \leftarrow \text{PreMul}([w], \max(N, M))$.
 8. Set $[y_P] = p_0 + \sum_{i=1}^N p_i[z_i]$.
 9. Set $[y_Q] = q_0 + \sum_{i=1}^M q_i[z_i]$.
 10. Compute $[y] \leftarrow \text{Div}([y_P], [y_Q])$.
 11. If $([s])$ then $[b] = 0 - [x]$ else $[b] = [x]$.
 12. Compute and return $[b] = [b] \cdot [y]$.
-

Recall that we recommend using a rational function in the form of $P(x)/Q(x)$ as in the above protocol for GCs-based implementation of high precision. With SS, we modify the protocol to evaluate only a single polynomial P of (a different) degree N and skip the division operation in step 10. Also, based on our calculations, the single polynomial variant is also slightly faster in the GC setting when precision is not high (e.g., with 16 bits of precision).

As far as optimizations go, we, as usual, simultaneously execute independent operations in the SS setting. Also, note that because coefficients p_i 's and q_j 's are not integers, evaluations of polynomials in steps 8 and 9 is not local in the SS setting and requires truncation. We, however, can reduce the cost of truncating N (respectively, M) values to the cost of one truncation. This is because we first add the products and truncate the sum once. This also applies to polynomial evaluation in other protocols such as cosine and arctangent.

When input a is known to lie in the range $[0, 90]$ (as in the fingerprint algorithms we consider in this work), steps 1, 2, 3, 11, and 12 are not executed and instead we return $[y] \cdot [x]$ after evaluating step 10. Lastly, when input a is known to be an integer, the comparisons in steps 1 and 3 and the multiplication of $[a]$ and a fixed-point constant $\frac{1}{90}$ in step 4 become more efficient in both SS and GCs-based implementations (i.e., both settings benefit from the reduced bitlength of a if only integer part is stored and the cost of the multiplication in the SS setting becomes 1 interactive operation).

To show security of this protocol, we need to build simulators according to Definition 1. The main argument here is that because we only use secure building blocks that do not reveal any private information, we apply Canetti's composition theorem [19] to result in security of the overall construction. More precisely, to simulate the adversarial view, we invoke simulators corresponding

to the building blocks. In more detail, if we assume an implementation based on a (n, τ) -threshold linear secret sharing, our protocols inherit the same security guarantees as those of the building blocks (i.e., perfect or statistical security in the presence of secure channels between the parties with at most τ corrupt computational parties) because no information about private values is revealed throughout the computation. More formally, to comply with the security definition, it is rather straightforward to build a simulator for our protocols by invoking simulators of the corresponding building blocks to result in the environment that will be indistinguishable from the real protocol execution by the participants. The same argument applies to other protocols presented in this work and we do not include explicit analysis (with the exception of selection).

The cosine function is evaluated similarly to sine. The main difference is in the way the input is pre- and post-processed for polynomial evaluation due to the behavior of this function. When cosine is evaluated using a rational function, we have the following secure protocol:

$[b] \leftarrow \text{Cos}([a])$

1. If $(\text{LT}(180, [a]))$ then $[a] = 360 - [a]$.
2. $[s] = \text{LT}(90, [a])$.
3. If $([s])$ then $[a] = 180 - [a]$.
4. Compute $[x] = \frac{1}{90}[a]$ and then $[x] = [x]^2$.
5. Lookup the minimum polynomial degrees N and M using $\beta = 1/2$ for which precision of the approximation is at least k bits.
6. Lookup polynomial coefficients p_0, \dots, p_N and q_0, \dots, q_M for cosine approximation.
7. Compute $([z_1], \dots, [z_{\max(N, M)}]) \leftarrow \text{PreMul}([x], \max(N, M))$.
8. Set $[y_P] = p_0 + \sum_{i=1}^N p_i [z_i]$.
9. Set $[y_Q] = q_0 + \sum_{i=1}^M q_i [z_i]$.
10. Compute $[y] \leftarrow \text{Div}([y_P], [y_Q])$.
11. If $([s])$ then $[b] = 0 - [y]$ else $[b] = [y]$.
12. Return $[b]$.

The same optimizations as those described for the sine protocol apply to the cosine computation as well. Furthermore, to use a single polynomial to evaluate cosine, we similarly lookup coefficients for a single polynomial of (a different) degree N and skip steps 9 and 10 in the `Cos` protocol.

The complexities of `Sin` and `Cos` are provided in Table 2 . To achieve 16, 32, or 64 bits of precision, N needs to be set to 3, 5, or 9, respectively, for both `Sin` and `Cos` using a single polynomial, and (N, M) need to be set to (2, 1), (3, 2), and (6, 2), respectively, for `Sin` using the rational function approach.

In the case of inverse trigonometric functions and arctangent in particular, the function input domain is the entire $(-\infty, \infty)$ and the range of output is $(-\pi/2, \pi/2)$. We recall that arctangent is an odd function (i.e., $\arctan(-x) = -\arctan(x)$) and for all positive inputs we have $\arctan(x) + \arctan(\frac{1}{x}) = \pi/2$. It is, therefore, sufficient to approximate arctangent over the interval of $[0, 1]$. To this end, we use the technique introduced by Medina in [43] and its formal proof in [27]. In particular, [43] defines a sequence of polynomials over the input domain of $[0, 1]$, denoted as $h_N(x)$, with the property that

$$|h_N(x) - \arctan(x)| \leq \left(\frac{1}{4^{5/8}} \right)^{\deg(h_N)+1}.$$

We use this formula to determine the degree N for any k -bit precision. This degree N is logarithmic with respect to the desired precision. Afterwards, the coefficients of $h_N(x)$ are computed from the recursive definitions in [43]. We choose this approach over other alternatives such as [30] for its efficiency. As an example, in [30] the authors propose to divide $[0, \infty)$ into $s + 1$ segments for some

Prot.	Secret sharing		Garbled circuits	
	Rounds	Interactive operations	XOR gates	Non-XOR gates
Sin	$2 \log N + 16$	$2Nk + 8\ell + 2N + 6k + 4$	$(\max(N, M) + N + M + 2) \times (4\ell^2 - 4\ell) + 7\ell^2 + 4\ell(N + M) + 31\ell$	$(\max(N, M) + N + M + 2) \times (2\ell^2 - \ell) + 3\ell^2 + \ell(N + M) + 11\ell$
Cos	$2 \log N + 15$	$2Nk + 8\ell + 2N + 4k + 2$	$(\max(N, M) + N + M + 1) \times (4\ell^2 - 4\ell) + 7\ell^2 + 4\ell(N + M) + 31\ell$	$(\max(N, M) + N + M + 1) \times (2\ell^2 - \ell) + 3\ell^2 + \ell(N + M) + 11\ell$
Arctan	$2 \log N + 3 \log \ell + 2 \log(\frac{\ell}{3.5}) + 22$	$1.5\ell \log \ell + 2\ell \log(\frac{\ell}{3.5}) + 2Nk + 18.5\ell + 2N + 4 \log(\frac{\ell}{3.5}) + 6$	$8N\ell^2 + 3\ell^2 - 4N\ell + 43\ell$	$4N\ell^2 + \ell^2 - N\ell + 15\ell$
Sqrt	$0.5\ell + 2 \log \ell + 6\xi + 24$	$2\ell^2 + \ell \log \ell + 3k(\xi + 1) + 5\ell + 6\xi + 12$	$12\xi\ell^2 + 12.5\ell^2 - k^2 + \ell k - 8\xi\ell - 7.5\ell + k - 2$	$6\xi\ell^2 + 6.5\ell^2 - k^2 + \ell k - 2\xi\ell - 0.5\ell + k - 4$
Select	$c(2 \log^2 t_1 + 2 \log^2 t_2 + 6 \log m + 2 \log \log m + 17)$	$c(\ell(t_1 - 0.25)(\log^2 t_1 + \log t_1 + 4) + \ell(t_2 - 0.25)(\log^2 t_2 + \log t_2 + 4) + 2m \log m \log \log m + 8m \log m + 18m\ell - 5.5m + 2 \log m - 8\ell + 9)$	$c(1.5t_1\ell(\log^2 t_1 + \log t_1 + 4) + 1.5t_2\ell(\log^2 t_2 + \log t_2 + 4) + (2\ell + 20.5)m \log m + m + 12.5m\ell - 6 \log m - 8\ell)$	$c(0.5t_1\ell(\log^2 t_1 + \log t_1 + 4) + 0.5t_2\ell(\log^2 t_2 + \log t_2 + 4) + (4\ell + 5)m \log m + m + 4m\ell + (2\ell - 1) \log m)$

Table 2: Performance of proposed secure building blocks for fixed-point values.

value of s and perform a search to discover in which interval the input falls. Afterwards, the input is transformed into another variable (involving division) whose value is guaranteed to be within a small fixed interval and arctangent of this new variable is approximated using standard polynomial approximation techniques. In this approach, the search can be performed using binary search trees whose secure realization is non-trivial or alternatively will involve s steps. This extra computation makes the overall performance of the solution in [30] worse than that of [43]’s approach used here. Another candidate for arctangent evaluation is the well-known Taylor series of arctangent. However, Taylor series provides possibly reasonable precision if the input is bound to be very close to one point from the input domain, which we cannot assume. In addition, arctangent Taylor series converges very slowly. For example, for a three decimal precision on inputs around 0.95 Taylor series requires a polynomial of degree 57, whereas [43]’s approach requires a 7th-degree polynomial [27]. Our secure protocol to approximate arctangent based on the approach from [43] is given next:

$[b] \leftarrow \text{Arctan}([a])$

1. Compute $[s] \leftarrow \text{LT}([a], 0)$.
 2. If $([s])$ then $[x] = 0 - [a]$ else $[x] = [a]$.
 3. Compute $[c] \leftarrow \text{LT}(1, [x])$.
 4. If $([c])$ then $[d] = \pi/2$, $[y] \leftarrow \text{Div}(1, [x])$; else $[d] = 0$, $[y] = [x]$.
 5. Lookup the minimum polynomial degree N for which precision of the approximation is at least k bits.
 6. Lookup polynomial coefficients p_0, \dots, p_N for arctangent approximation from [43].
 7. Compute $([z_1], \dots, [z_N]) \leftarrow \text{PreMul}([y], N)$.
 8. Set $[z] = p_0 + \sum_{i=1}^N p_i [z_i]$.
 9. If $([c])$ then $[z] = [d] - [z]$ else $[z] = [d] + [z]$.
 10. If $([s])$ then $[b] = 0 - [z]$ else $[b] = [z]$.
 11. Return $[b]$.
-

The complexity of this protocol can be found in Table 2 .

5.2.2 Square Root

We now proceed with the square root computation defined by the interface $[b] \leftarrow \text{Sqrt}([a])$, where a and b are fixed-point values to cover the general case.

Secure multi-party computation of the square root based on SS has been treated by Liedel in [40]. The approach is based on the Goldschmidt's algorithm, which is faster than the Newton-Raphson's method. However, to eliminate the accumulated errors, the last iteration of the algorithm is replaced with the self-correcting iteration of the Newton-Raphson's method. For an ℓ -bit input a with k bits after the radix point, this protocol uses $2\ell^2 + \ell \log \ell + 3k(\xi + 1) + 5\ell + 6\xi + 12$ interactive operations in $0.5\ell + 2 \log \ell + 6\xi + 24$ rounds, where ℓ is assumed to be a power of 2 and $\xi = \lceil \log_2(\ell/5.4) \rceil$ is the number of algorithm iterations. For the purposes of this work, we use the protocol from [40] for the SS setting. We also optimize the computation to be used with GCs based on the specifics of that technique as described next. Furthermore, because polynomial approximation (similar to the way it was done for trigonometric functions in section 5.2.1) is attractive in the SS setting, we provide an alternative solution in Appendix A.2.

Goldschmidt's method starts by computing an initial approximation for $\frac{1}{\sqrt{a}}$, denoted by b_0 , that satisfies $\frac{1}{2} \leq ab_0^2 < \frac{3}{2}$. It then proceeds in iterations increasing the precision of the approximation with each consecutive iteration. To approximate $\frac{1}{\sqrt{a}}$, [40] uses an efficient method (a linear equation) that expects that input a is in the range $[\frac{1}{2}, 1)$. Thus, there is a need to first normalize a to a' such that $\frac{1}{2} \leq a' < 1$ and $a = a' \cdot 2^w$. Note that $\frac{1}{\sqrt{a}} = \frac{1}{\sqrt{a'}} \sqrt{2^{-w}}$, therefore, once we approximate $\frac{1}{\sqrt{a'}}$, we can multiply it by $\sqrt{2^{-w}}$ to determine an approximation of $\frac{1}{\sqrt{a}}$.

In our (ℓ, k) -bit fixed-point representation, a normalized input $a' \in [\frac{1}{2}, 1)$ has the most significant non-zero bit exactly at position $k - 1$. We also express $\sqrt{2^{-w}}$ as $\frac{1}{\sqrt{2}} 2^{-\lfloor \frac{w}{2} \rfloor}$ when w is odd and as $2^{-\lfloor \frac{w}{2} \rfloor}$ when w is even. Our normalization procedure Norm that we present next thus takes input a and returns $\frac{1}{2} \leq a' < 1$ (normalized input a in (ℓ, k) -bit fixed-point representation), $2^{-\lfloor \frac{w}{2} \rfloor}$ and bit c set to 1 when w is odd and to 0 otherwise. The protocol is optimized for the GC approach with cheap XOR gates [38]. It assumes that ℓ and k are even.

$\langle [a'], [2^{-\lfloor \frac{w}{2} \rfloor}], [c] \rangle \leftarrow \text{Norm}([a])$

1. $([a_{\ell-1}], \dots, [a_0]) \leftarrow [a]$.
 2. Set $[x_{\ell-1}] = [a_{\ell-1}]$.
 3. For $i = \ell - 2, \dots, 0$ do $[x_i] = [a_i] \vee [x_{i+1}]$.
 4. Set $[y_{\ell-1}] = [x_{\ell-1}]$.
 5. For $i = 0, \dots, \ell - 2$ do in parallel $[y_i] = [x_i] \oplus [x_{i+1}]$.
 6. For $i = 0, \dots, \ell - 1$ do in parallel $[z^{(i)}] \leftarrow ([a_{\ell-1}] \wedge [y_i], \dots, [a_0] \wedge [y_i])$.
 7. Compute $[a'] = \left(\bigoplus_{i=0}^{k-1} ([z^{(i)}] \ll (k - 1 - i)) \right) \oplus \left(\bigoplus_{i=k}^{\ell-1} ([z^{(i)}] \gg (i - (k - 1))) \right)$.
 8. Let $[u_0] = [y_0]$, $[u_{\frac{\ell}{2}}] = [y_{\ell-1}]$ and for $i = 1, \dots, \frac{\ell}{2} - 1$ do in parallel $[u_i] = [y_{2i-1}] \oplus [y_{2i}]$.
 9. Set $[2^{-\lfloor \frac{w}{2} \rfloor}] \leftarrow ([d_{\ell-1}], \dots, [d_0]) = (0^{\ell - \frac{3k}{2} - 1}, [u_0], [u_1], \dots, [u_{\frac{\ell}{2}}], 0^{\frac{3k-\ell}{2}})$, where 0^x corresponds to x zeros.
 10. Set $[c] = \bigoplus_{i=0}^{\frac{\ell}{2}-1} [y_{2i}]$.
 11. Return $\langle [a'], [2^{-\lfloor \frac{w}{2} \rfloor}], [c] \rangle$.
-

Here, lines 2–3 preserve the most significant zero bits of a and set the remaining bits to 1 in variable x (i.e., all bits following the most significant non-zero bit are 1). Lines 4–5 compute y as a vector with the most significant non-zero bit of a set to 1 and all other bits set to 0. On line 6, each vector $z^{(i)}$ is either filled with 0s or set to a depending on the i th bit of y (thus, all but one $z^{(i)}$ can be non-zero). Line 7 computes the normalized value of a by aggregating all vectors $z^{(i)}$ shifted an

appropriate number of positions. Here operation $x \ll y$ shifts ℓ -bit representation of x y positions to the left by discarding y most significant bits of x and appending y 0s in place of least significant bits. Similarly, $x \gg y$ shifts x to the right by prepending y 0s in place of most significant bits and discarding y least significant bits of x . Note that we can use cheap XOR gates for this aggregation operation because at most one y_i can take a non-zero value.

Lines 8 and 9 compute $[2^{-\lfloor \frac{w}{2} \rfloor}]$. Because a pair of consecutive i 's results in the same value of w , we first combine the pairs on line 8 and shift them in place on line 9. As before, we can use cheap XOR and free shift operations to accomplish this task because at most one y_i is set. Lastly, line 10 computes the bit c , which is set by combining all flags y_i 's at odd distances from $k - 1$.

Note that for simplicity of exposition, we AND and XOR all bits in steps 6 and 7. There is, however, no need to compute the AND for the bits discarded in step 7 or compute the XOR with newly appended or prepended 0s. We obtain that this protocol can be implemented as a circuit using $0.5\ell^2 - k^2 + \ell k + 1.5\ell - 4$ non-XOR and $0.5\ell^2 - k^2 + \ell k + 0.5\ell - 3$ XOR gates.

Once we determine normalized input a' , Liedel [40] approximates $\frac{1}{\sqrt{a'}}$, denoted by b'_0 , by a linear equation $b'_0 = \alpha a' + \beta$, where α and β are precomputed. The values of these coefficients are set to $\alpha = -0.8099868542$ and $\beta = 1.787727479$ in [40] to compute an initial approximation b'_0 with almost 5.5 bits of precision (when a' is in the range $[\frac{1}{2}, 1)$). We then use b'_0 to compute b_0 that approximates $\frac{1}{\sqrt{a}}$ as described above.

After the initialization, Goldschmidt's algorithm sets $g_0 = xb_0$ and $h_0 = 0.5b_0$ and proceeds in ξ iterations that compute: $g_{i+1} = g_i(1.5 - g_i h_i)$, $h_{i+1} = h_i(1.5 - g_i h_i)$. The last iteration is replaced with one iteration of Newton-Raphson's method to eliminate accumulated errors that computes the following: $h_{i+1} = h_i(1.5 - 0.5ah_i^2)$. This gives us the following square root protocol, which we present optimized for the GC approach.

$[b] \leftarrow \text{Sqrt}([a])$

1. Let $\xi = \lceil \log_2(\frac{\ell}{5.4}) \rceil$.
 2. Execute $\langle [a'], [2^{-\lfloor \frac{w}{2} \rfloor}], [c] \rangle \leftarrow \text{Norm}([a])$.
 3. Let $\alpha = -0.8099868542$, $\beta = 1.787727479$ and compute $[b'_0] = \alpha \cdot [a'] + \beta$.
 4. Compute $[b_0] = \left(\left([c] \wedge \frac{1}{\sqrt{2}} \right) \oplus \neg[c] \right) \cdot [2^{-\lfloor \frac{w}{2} \rfloor}] \cdot [b'_0]$.
 5. Compute $[g_0] = [a] \cdot [b_0]$ and $[h_0] = ([b_0] \gg 1)$.
 6. For $i = 0, \dots, \xi - 2$ do
 - (a) $[x] = 1.5 - [g_i] \cdot [h_i]$.
 - (b) if $(i < \xi - 2)$ then $[g_{i+1}] = [g_i] \cdot [x]$.
 - (c) $[h_{i+1}] = [h_i] \cdot [x]$.
 7. Compute $[h_\xi] = [h_{\xi-1}](1.5 - ([a] \cdot [h_{\xi-1}]^2 \gg 1))$.
 8. Return $[b] = [h_\xi]$.
-

Here multiplication by 0.5 is replaced by shift to the right by 1 bit that has no cost. Complexity of this protocol can be found in Table 2 .

5.2.3 Selection

There are multiple ways to find the f th smallest element of a set. The most straightforward method is to obviously sort the input set and return the f th element of the sorted set. This approach is often beneficial when the input set is small. It is usually faster to use a selection algorithm, and the goal of this section is to design an oblivious selection protocol for finding the f th smallest element of a set.

The regular non-oblivious selection algorithms run in $O(m)$ time on m -element sets. We con-

sidered both deterministic and probabilistic algorithms and settled on a probabilistic algorithm that performs the best in the secure setting in practice. Its complexity is $O(m \log m)$ due to the use of data-oblivious compaction. Goodrich [29] proposed a probabilistic oblivious selection algorithm that works in linear time. Our starting point, however, is a different simpler algorithm that performs well in practice. Our algorithm proceeds by scanning through the input set and selects each element with probability $\frac{c_1}{\sqrt{m}}$ for some constant c_1 . The expected number of selected elements is $O(\sqrt{m})$. The algorithm then sorts the selected elements and determines two elements x and y between which the f th smallest element is expected to lie based on f , m , and the number of selected elements. Afterwards, we scan the input set again retrieving all elements with values between x and y and simultaneously computing the ranks of x and y in the input set; let r_x and r_y denote the ranks. The expected number of retrieved elements is $O(\sqrt{m})$. We sort the retrieved elements and return the element at position $f - r_x$ in the sorted set. Let t_1 denote the number of elements randomly selected in the beginning of the algorithm and t_2 denote the number of elements with values between x and y .

Because of the randomized nature of our algorithm, it can fail at multiple points of its execution. In particular, if the number of elements selected in the beginning of the algorithm is too large or too small (and the performance guarantees cannot be met), we abort. Similarly, if the number of the retrieved elements that lie between x and y is too small, we abort. Lastly, if f does not lie between the ranks of x and y , the f th smallest element will not be among the retrieved elements and we abort. It can be shown using the union bound that all of the above events happen with a probability negligible in the input size. By using the appropriate choice of constants we can also ensure that faults are very rare in practice (see below). Thus, in the rare event of failure, the algorithm needs to be restarted using new random choices, and the expected number of times the algorithm is to be executed is slightly above 1.

Our protocol **Select** is given below. Compared to the algorithm structure outlined above, we need to ensure high probability of success through the appropriate selection of x and y as well as some constant parameters. We also need to ensure that the execution can proceed obliviously without compromising private data. To achieve this, we use three constants c_1 , c_2 , and \hat{c} . The value of c_1 influences the probability with which an element is selected in the beginning of the algorithm. The value of \hat{c} influences the distance between x and y in the set of t_1 selected elements. In particular, we first determine the position where f is expected to be positioned among the t_1 selected elements as $(ft_1)/m$. We then choose x to be \hat{c} elements before that position and choose y to be $3\hat{c}$ elements after that position. Lastly, c_2 is used to control the size of t_2 . When t_2 is too small due to the random choices made during the execution, we abort.

To turn this logic into a data-oblivious protocol, we resort to secure and oblivious set operations. In particular, after we mark each element of the input set as selected or not selected, we use data-oblivious compaction to place all selected elements in the beginning of the set. Similarly, we use data-oblivious sorting to sort t_1 and t_2 elements.

$[b] \leftarrow \text{Select}(\langle [a_1], \dots, [a_m] \rangle, f)$

1. Set constants c_1 , c_2 , and \hat{c} ; also set $n = \sqrt{m}$, $[t_1] = 0$, and $[t_2] = 0$.
2. For $i = 1, \dots, m$ set $[b_i] = 0$.
3. For $i = 1, \dots, m$ do in parallel
 - (a) Generate random $[r_i] \in \mathbb{Z}_n$.
 - (b) If $\text{LT}([r_i], c_1)$ then $[b_i] = 1$.
4. For $i = 1, \dots, m$ do $[t_1] = [t_1] + [b_i]$.
5. Open the value of t_1 .
6. If $((t_1 > 2c_1n) \vee (t_1 < \frac{1}{2}c_1n))$ then abort.

7. Execute $\langle [a'_1], \dots, [a'_m] \rangle \leftarrow \text{Comp}(\langle [b_1], [a_1] \wedge [b_1] \rangle, \dots, \langle [b_m], [a_m] \wedge [b_m] \rangle)$.
8. Execute $\langle [a''_1], \dots, [a''_{t_1}] \rangle \leftarrow \text{Sort}([a'_1], \dots, [a'_{t_1}])$.
9. Compute k as the closest integer to $(ft_1)/m$.
10. If $(k - \hat{c} < 1)$ then $[x] \leftarrow \text{Min}([a_1], \dots, [a_m])$; else $[x] = [a''_{k-\hat{c}}]$.
11. If $(k + 3\hat{c} > t_1)$ then $[y] \leftarrow \text{Max}([a_1], \dots, [a_m])$; else $[y] = [a''_{k+3\hat{c}}]$.
12. Set $[r_x] = 0$ and $[r_y] = 0$.
13. For $i = 1, \dots, m$ do in parallel $[g_i] = \text{LT}([a_i], [x])$ and $[g'_i] = \text{LT}([a_i], [y])$.
14. For $i = 1, \dots, m$ do in parallel $[b'_i] = \neg[g_i] \wedge [g'_i]$.
15. For $i = 1, \dots, m$ do $[r_x] = [r_x] + [g_i]$ and $[t_2] = [t_2] + [b'_i]$.
16. Open the values of r_x and t_2 .
17. If $((t_2 < 4\hat{c}c_2n) \vee ((f - r_x) < 0) \vee ((f - r_x) > t_2))$ then abort.
18. Execute $\langle [d_1], \dots, [d_m] \rangle \leftarrow \text{Comp}(\langle [b'_1], [a_1] \wedge [b'_1] \rangle, \dots, \langle [b'_m], [a_m] \wedge [b'_m] \rangle)$.
19. Execute $\langle [d'_1], \dots, [d'_{t_2}] \rangle \leftarrow \text{Sort}([d_1], \dots, [d_{t_2}])$.
20. Return $[b] = [d'_{f-r_x}]$.

Note that because maximum and minimum functions are evaluated on the same set, we reduce the overhead of evaluating them simultaneously compared to evaluating them individually. In particular, when min and max functions are evaluated in a tree-like fashion on an m -element set, we use the $m/2$ comparison in the first layer to set both minimum and maximum elements of each pair using the same cost as evaluating only one of them. The rest is evaluated as before, and we save about $1/2$ overhead of one of min or max.

Complexity of this protocol is given in Table 2 as a function of parameters m , t_1 , t_2 , c , and bitlength ℓ . The expected value of t_1 is $c_1\sqrt{m}$, the expected value of t_2 is $(4\hat{c}\sqrt{m})/c_1$, and c indicates the average number of times the algorithm needs to be invoked, which is a constant slightly larger than 1 (see below).

We experimentally determined the best values for c_1 , c_2 , and \hat{c} for different values of n to ensure high success of the algorithm. Based on the experiments, we recommend to set $c_1 = 2$, $c_2 = 2$, and $\hat{c} = 10$. Using these values, the probability of the protocol's failure on a single run is at most a few percent up to input size 10,000. With larger input sizes, a larger value of \hat{c} might be preferred.

Unlike all other protocols presented in this work, three values t_1 , t_2 , and r_x privately computed in **Select** are opened during its execution. Thus, to guarantee security, we need to show that these values are independent of the private input set and result in no information leakage. First, observe that the value of t_1 is computed purely based on random choices made in step 3(a) of the protocol. Thus, its value is data-independent. Second, the rank of x r_x is determined by random choices, but not the data values themselves. Similarly, the value of t_2 depends only on the ranks of the randomly chosen pivots, but not on the actual data values. Therefore, execution is data-oblivious and security is maintained.

6 Secure Fingerprint Recognition

We are now ready to proceed with the description of our solutions for secure fingerprint recognition using the building blocks introduced in Section 5. We provide three constructions, one for each fingerprint recognition approach described in Section 3.

6.1 Secure Fingerprint Recognition using Brute Force Geometrical Transformation

The easiest way to execute all (non-secure) fingerprint recognition algorithms in Section 3 is to use floating-point representation. We, however, choose to utilize integer and fixed-point arithmetic

in their secure counterparts to reduce overhead associated with secure execution. In particular, typically minutia coordinates (x_i, y_i) as well as their orientation θ_i are represented as integers. This means that all inputs in Algorithm 1, the first fingerprint matching algorithm based on brute force geometrical transformation, are integers and computing the number of matched minutiae using Algorithm 2 can also be performed on integers. The output of sine and cosine functions in step 2(b) of Algorithm 1, however, are not integers and we utilize fixed-point representation for their values. Moreover, after the minutia points are transformed, we can truncate the transformed coordinates x_i'' and y_i'' and use their integer representation in Algorithm 2.

Our secure implementation of Algorithm 1 is given below as protocol **GeomTransFR**. The secure computation uses the same logic as Algorithm 1 with the difference that the computation of the maximum matching is performed outside the main for-loop to reduce the number of rounds in the SS setting.

$([C_{max}], \langle [\Delta x_{max}], [\Delta y_{max}], [\Delta \theta_{max}] \rangle) \leftarrow \text{GeomTransFR } (T = \{t_i = ([x_i], [y_i], [\theta_i])\}_{i=1}^m, S = \{s_i = ([x'_i], [y'_i], [\theta'_i])\}_{i=1}^n)$

1. $[C_{max}] = [0]$;
 2. For $i = 1, \dots, m$ and $j = 1, \dots, n$, compute in parallel:
 - (a) $[\Delta x_{i,j}] = [x'_j] - [x_i]$, $[\Delta y_{i,j}] = [y'_j] - [y_i]$, and $[\Delta \theta_{i,j}] = [\theta'_j] - [\theta_i]$.
 - (b) $[c_{i,j}] = \text{Sin}([\Delta \theta_{i,j}])$ and $[c'_{i,j}] = \text{Cos}([\Delta \theta_{i,j}])$.
 - (c) For $k = 1, \dots, n$, compute in parallel $[x_{i,j}^{(k)}] = [c'_{i,j}] \cdot [x'_k] + [c_{i,j}] \cdot [y'_k] - [\Delta x_{i,j}]$, $[y_{i,j}^{(k)}] = [c'_{i,j}] \cdot [y'_k] - [c_{i,j}] \cdot [x'_k] - [\Delta y_{i,j}]$, and $[\theta_{i,j}^{(k)}] = [\theta'_k] - [\Delta \theta_{i,j}]$ and save the computed points as $S_{i,j} = \{([x_{i,j}^{(k)}], [y_{i,j}^{(k)}], [\theta_{i,j}^{(k)}])\}_{k=1}^n$.
 - (d) Compute the number $[C_{i,j}]$ of matched minutiae between T and $S_{i,j}$ using protocol **Match**.
 3. Compute the largest matching $\langle [C_{max}], [\Delta x_{max}], [\Delta y_{max}], [\Delta \theta_{max}] \rangle = \text{Max}(\langle [C_{1,1}], [\Delta x_{1,1}], [\Delta y_{1,1}], [\Delta \theta_{1,1}] \rangle, \dots, \langle [C_{m,n}], [\Delta x_{m,n}], [\Delta y_{m,n}], [\Delta \theta_{m,n}] \rangle)$.
 4. Return $[C_{max}]$ and the corresponding alignment $\langle [\Delta x_{max}], [\Delta y_{max}], [\Delta \theta_{max}] \rangle$.
-

We obtain that all steps use integer arithmetic except step 2(b) (where sine and cosine have integer inputs, but fixed-point outputs) and multiplications in step 2(c) take one integer and one fixed-point operand (after which $x_{i,j}^{(k)}$ and $y_{i,j}^{(k)}$ are truncated to integer representation). Note that the latter corresponds to some optimization of the computation, where instead of converting integers x'_k and y'_k to fixed-point values and multiplying two fixed-point numbers, we can reduce the overhead by multiplying an integer to a fixed-point value. This eliminates the cost of truncating the product in the SS setting and reduces the cost of multiplication in the GC setting. Furthermore, we can also optimize the point at which the fixed-point values are converted back to integers in the computation of $x_{i,j}^{(k)}$ and $y_{i,j}^{(k)}$ in step 2(c) of the algorithm. In particular, in the SS setting we add the fixed-point products and $\Delta x_{i,j}^{(k)}$, $\Delta y_{i,j}^{(k)}$ converted to fixed-point representation (all of which have 0 cost), after which the sum is converted to an integer (paying the price of truncation). In the GCs setting, on the other hand, we could convert the products to integers first (which has 0 cost) and then perform addition/subtraction on shorter integer values.

Our secure implementation of Algorithm 2 is given as protocol **Match**. All computation is performed on integer values. Compared to the structure of Algorithm 2, there are a few notable differences. First, note that instead of checking the $\sqrt{(x_i - x'_j)^2 + (y_i - y'_j)^2} < \lambda$ constraint, the protocol checks the equivalent constraint $(x_i - x'_j)^2 + (y_i - y'_j)^2 < \lambda^2$ to avoid using a costly square

root operation (and it is assumed that λ^2 is supplied as part of the input instead of λ). Second, to evaluate the constraint $\min(|\theta_i - \theta'_j|, 360 - |\theta_i - \theta'_j|) < \lambda_\theta$, instead of computing $|\theta_i - \theta'_j|$ the protocol computes $\theta_i - \theta'_j$ or $\theta'_j - \theta_i$ depending on which value is positive and uses that value in the consecutive computation. In addition, instead of computing the minimum, the computation proceeds as $(|\theta_i - \theta'_j| < \lambda_\theta) \vee (360 - |\theta_i - \theta'_j| < \lambda_\theta)$ in steps 3(c) and 3(d) of the protocol, which allows us to make the computation slightly faster. Because this computation is performed a large number of times, even small improvements can have impact on the overall performance.

Similar to restructuring protocol **GeomTransFR** to maximize parallelism and lower the number of rounds, in **Match** all distance and orientation constraints are evaluated in parallel in step 3. Then step 4 iterates through all minutiae in T and constructs a matching between a minutia of T and an available minutia from S within a close proximity to it (if any). Variable l_j indicates whether the j th minutia of fingerprint S is currently available (i.e., has not yet been paired up with another minutia from T). For each minutia, s_j , marked as unavailable, its distance to the i th minutia in T is set to λ^2 to prevent it from being chosen for pairing the second time. Because step 4(d) is written to run all loop iterations in parallel, there is a single variable C_j for each loop iteration, all values of which are added together at the end of the loop (which is free using SS). With GCs this parallelism is often not essential, and if the loop is executed sequentially, C can be incremented on line 4(d) directly instead of using C_j 's. (And if parallel computation is used, the sum on line 4(e) will need to be implemented as the (free) XOR of all C_j .)

$[C] \leftarrow \text{Match}(T = \{t_i = ([x_i], [y_i], [\theta_i])\}_{i=1}^m, S = \{s_i = ([x'_i], [y'_i], [\theta'_i])\}_{i=1}^n, \lambda^2, \lambda_\theta)$

1. Set $[C] = [0]$.
 2. For $j = 1, \dots, n$, set in parallel $[l_j] = [0]$.
 3. For $i = 1, \dots, m$ and $j = 1, \dots, n$, compute in parallel:
 - (a) $[d_{i,j}] = ([x_i] - [x'_j])^2 + ([y_i] - [y'_j])^2$.
 - (b) If $\text{LT}([\theta_i], [\theta'_j])$, then $[a_{i,j}] = [\theta'_j] - [\theta_i]$, else $[a_{i,j}] = [\theta_i] - [\theta'_j]$.
 - (c) $[c_{i,j}] = \text{LT}([d_{i,j}], [\lambda^2])$, $[c'_{i,j}] = \text{LT}([a_{i,j}], [\lambda_\theta])$, and $[c''_{i,j}] = \text{LT}((360 - [a_{i,j}]), [\lambda_\theta])$.
 - (d) $[v_{i,j}] = [c_{i,j}] \wedge ([c'_{i,j}] \vee [c''_{i,j}])$.
 4. For $i = 1, \dots, m$, do
 - (a) For $j = 1, \dots, n$, do in parallel: if $([l_j] \vee \neg[v_{i,j}])$, then $[d_{i,j}] = \lambda^2$.
 - (b) Execute $([d_{\min}], [j_{\min}]) = \text{Min}([d_{i,1}], \dots, [d_{i,n}])$.
 - (c) Set $[u] = \text{LT}([d_{\min}], \lambda^2)$.
 - (d) For $j = 1, \dots, n$, compute in parallel: if $(\text{EQ}([j_{\min}], j) \wedge [u])$, then $[C_j] = [1]$ and $[l_j] = [1]$, else $[C_j] = [0]$.
 - (e) $[C] = [C] + \sum_{j=1}^n [C_j]$.
 5. Return $[C]$.
-

The asymptotic complexity of **GeomTransFR** and **Match** remains similar to their original non-secure counterparts. In particular, if we treat the bitlength of integer and fixed-point values as constants, the complexity of **GeomTransFR** is $O(n^2 m^2)$ and **Match** is $O(nm)$. Their round complexity (for the SS setting) is $O(m \log n)$ and $O(m \log n)$, respectively. If we wish to include dependency on the bitlengths ℓ and k , the complexity increases by at most a factor of $O(\ell)$ in the SS setting and at most a factor of $O(\ell^2)$ in the GCs setting due to the differences in the complexity of the underlying building blocks.

6.2 Secure Fingerprint Recognition using High Curvature Points for Alignment

We next treat our secure realization of fingerprint recognition using high curvature points from Algorithm 3. The corresponding secure computation is provided as protocol **HighCurvatureFR** below. It makes calls to a secure version of Algorithm 4, which we consequently call as protocol

OptimalMotion. Also, because of the complexity of the computation associated with finding the closest points in step 3 of Algorithm 3, we provide the corresponding secure computation as a separate protocol **ClosestPoints**. All computation is performed on fixed-point values with the exception of step 7 of **HighCurvatureFR**, where a call to the minutia pairing protocol **Match** is made.

$([C], (R, v)) \leftarrow \text{HighCurvatureFR}(T = (\{t_i = ([x_i], [y_i], [\theta_i])\}_{i=1}^m, \{\hat{t}_i = ([\hat{x}_i], [\hat{y}_i], [\hat{w}_i])\}_{i=1}^{\hat{m}}), S = (\{s_i = ([x'_i], [y'_i], [\theta'_i])\}_{i=1}^n, \{\hat{s}_i = ([\hat{x}'_i], [\hat{y}'_i], [\hat{w}'_i])\}_{i=1}^{\hat{n}}, f, \gamma, (\alpha_1, \dots, \alpha_\gamma), \beta, \lambda^2, \lambda_\theta)$

1. Set $[S_{LTS}] = [0]$.
 2. For $i = 1, \dots, \hat{n}$, set $\bar{s}_i = \hat{s}_i$.
 3. For $\text{ind} = 1, \dots, \gamma$ do
 - (a) Execute $\{([d_i], \tilde{t}_i)\}_{i=1}^{\hat{n}} \leftarrow \text{ClosestPoints}(\{\hat{t}_i\}_{i=1}^{\hat{m}}, \{\hat{s}_i\}_{i=1}^{\hat{n}}, \alpha_{\text{ind}})$.
 - (b) Execute $[y] \leftarrow \text{Select}([d_1], \dots, [d_{\hat{n}}], f)$.
 - (c) For $i = 1, \dots, \hat{n}$ do in parallel $[l_i] = \text{LT}([d'_i], [y] + 1)$.
 - (d) Execute $([a_1], b_1, c_1), \dots, ([a_{\hat{n}}], b_{\hat{n}}, c_{\hat{n}}) \leftarrow \text{Comp}([l_1], \hat{s}_1 \wedge [l_1], \tilde{t}_1 \wedge [l_1], \dots, ([l_{\hat{n}}], \hat{s}_{\hat{n}} \wedge [l_{\hat{n}}], \tilde{t}_{\hat{n}} \wedge [l_{\hat{n}}]))$ using $[l_i]$'s as the keys.
 - (e) Compute the optimal motion $(R, v) \leftarrow \text{OptimalMotion}(\{c_i, b_i\}_{i=1}^f)$.
 - (f) For $i = 1, \dots, \hat{n}$ transform the points in parallel as $[x''_i] = [v_1] + [r_{11}] \cdot [\hat{x}'_i] + [r_{12}] \cdot [\hat{y}'_i] + [r_{13}] \cdot [\hat{w}'_i]$, $[y''_i] = [v_2] + [r_{21}] \cdot [\hat{x}'_i] + [r_{22}] \cdot [\hat{y}'_i] + [r_{23}] \cdot [\hat{w}'_i]$, and $[w''_i] = [v_3] + [r_{31}] \cdot [\hat{x}'_i] + [r_{32}] \cdot [\hat{y}'_i] + [r_{33}] \cdot [\hat{w}'_i]$, then set $\hat{s}_i = ([x''_i], [y''_i], [w''_i])$.
 4. Execute $(R, v) \leftarrow \text{OptimalMotion}(\{\bar{s}_i, \hat{s}_i\}_{i=1}^{\hat{n}})$.
 5. Compute $[c_1] = \text{Div}([\bar{y}_2] - [\bar{y}_1], [\bar{x}_2] - [\bar{x}_1])$, $[c_2] = \text{Div}([\hat{y}'_2] - [\hat{y}'_1], [\hat{x}'_2] - [\hat{x}'_1])$, $[c_3] = \text{Div}([c_1] - [c_2], 1 + [c_1] \cdot [c_2])$, and $[\Delta\theta] = \text{Arctan}([c_3])$.
 6. For $i = 1, \dots, n$ do in parallel $[x''_i] = [v_1] + [r_{11}] \cdot [x'_i] + [r_{12}] \cdot [y'_i]$ and $[y''_i] = [v_2] + [r_{21}] \cdot [x'_i] + [r_{22}] \cdot [y'_i]$, then set $s_i = ([x''_i], [y''_i], [\theta'_i] - [\Delta\theta])$.
 7. Compute the number $[C]$ of matched minutiae by executing **Match** $(\{t_i\}_{i=1}^m, \{s_i\}_{i=1}^n, \lambda^2, \lambda_\theta)$.
 8. Return $[C], (R, v)$.
-

Different from the computation in Algorithm 3, our protocol **HighCurvatureFR** proceeds using the maximum number of iterations γ , as not to reveal the actual number of iterations needed (which depends on private inputs). The remaining algorithm's structure is maintained, where the computation is replaced with secure and data-oblivious operations. In particular, in each iteration of step 3, we first determine the closest high-curvature point from T to each (possibly transformed) high-curvature point from S and compute f pairs with the smallest distances. Secure computation of the closest points using protocol **ClosestPoints** is described below, while determining the closest f pairs is performed on steps 3(b)–(d) as follows. After selecting the f th smallest element y among the computed distances (step 3(b)), each distance is compared to that element (step 3(c)). We then proceed with pushing all elements which are less than y to the beginning of the set using compaction (step 3(d)) and consequently use the f smallest distances (located in the beginning of the set) for optimal motion computation.

Once the closest f pairs and the corresponding optimal motion have been determined, the protocol proceeds with applying the optimal motion to the high-curvature points in S . After executing this computation a necessary number of times, the protocol computes the overall motion in step 4 and applies it to the minutia points in the same way as in Algorithm 3. One thing to notice is that **HighCurvatureFR** has additional (public) inputs compared to Algorithm 3. The parameters $\alpha_1, \dots, \alpha_\gamma$ specify bounding box sizes for the purposes of closest points computation in step 3(a) (see below). The remaining new parameters λ^2 , λ_θ and β are made explicit as inputs to the minutia pairing protocol **Match** and the scaling factor in distance computation (step 3 of Algorithm 3).

The protocol **ClosestPoints** below takes two sets of points t_i 's and s_i 's (represented using three coordinates each) and for each point s_i returns the closest element among the t_i 's and the corresponding distance. As mentioned in section 3.2, the TICP algorithm on which our construction

builds uses the bounding box approach to eliminate errors during this computation. In particular, only points within a certain distance from a point are considered, and the maximum allowable distance (i.e., the bounding box size) is denoted by α_i in the i th iteration of **HighCurvatureFR** or Algorithm 3. The sizes of the bounding boxes become smaller with each iteration as the two sets of points become closer to each other. When we make a call to **ClosestPoints**, we pass a single bounding box size for the current iteration and that parameter is denoted as α in the interface of **ClosestPoints**.

$$\{([d_i], \tilde{t}_i)\}_{i=1}^{\hat{n}} \leftarrow \text{ClosestPoints}(\{t_i = (x_i, y_i, w_i)\}_{i=1}^{\hat{m}}, \{s_i = (x'_i, y'_i, w'_i)\}_{i=1}^{\hat{n}}, \alpha, \beta)$$

1. For $i = 1, \dots, \hat{n}$ and for $j = 1, \dots, \hat{m}$ do in parallel:
 - (a) Set $[d_{(i,j)}] = \text{Sqrt}([\hat{x}_j] - [\hat{x}'_i])^2 + ([\hat{y}_j] - [\hat{y}'_i])^2$.
 - (b) If $(\text{LT}([\hat{w}_j], [\hat{w}'_i]))$ then $[a_{(i,j)}] = [\hat{w}'_i] - [\hat{w}_j]$; else $[a_{(i,j)}] = [\hat{w}_j] - [\hat{w}'_i]$.
 - (c) Set $[d_{(i,j)}] + \beta \cdot [a_{(i,j)}]$.
 2. For $i = 1, \dots, \hat{m}$ do in parallel $[l_i] = 1$.
 3. For $i = 1, \dots, \hat{n}$ do in parallel $[l'_i] = 1$.
 4. For $i = 1, \dots, \hat{n}$ do
 - (a) For $j = 1, \dots, \hat{m}$ do in parallel if $(\neg[l_j])$ then $[d_{(i,j)}] = \infty$.
 - (b) Execute $([d_i], [j_{\min}]) \leftarrow \text{Min}([d_{(i,1)}], \dots, [d_{(i,\hat{m})}])$.
 - (c) $[b] = \text{LT}([d_i], \alpha)$.
 - (d) For $j = 1, \dots, \hat{m}$ do in parallel if $(\text{EQ}([j_{\min}], j) \wedge [b])$ then $[l_j] = 0, [l'_i] = 0, \tilde{t}_i = t_j$.
 5. For $i = 1, \dots, \hat{n}$ do
 - (a) For $j = 1, \dots, \hat{m}$ do in parallel if $(\neg[l_j])$ then $[d_{(i,j)}] = \infty$.
 - (b) Execute $([d_i], [j_{\min}]) \leftarrow \text{Min}([d_{(i,1)}], \dots, [d_{(i,\hat{m})}])$.
 - (c) For $j = 1, \dots, \hat{m}$ do in parallel if $(\text{EQ}([j_{\min}], j) \wedge [l'_i])$ then $[l_j] = 0, [l'_i] = 0, \tilde{t}_i = t_j$.
 6. Return $\{([d_i], \tilde{t}_i)\}_{i=1}^{\hat{n}}$.
-

Given two sets of points and public parameter α , the **ClosestPoints** protocol first computes the distances between each pair of points in parallel in step 1 (according to the formula in step 3 of Algorithm 3). Next, we mark each t_i and s_i as available (steps 2 and 3, respectively). Step 4 iterates through all s_i 's and determines the closest available point t_j to s_i (the distance to the unavailable points is set to infinity to ensure that they are not chosen). If the closest point is within the bounding box, s_i is paired up with t_j and both are marked as unavailable.

At the end of step 4, some points s_i 's will be paired up with one of the t_j 's, while others will not be. To ensure that the algorithm produces enough pairs for their consecutive use in **HighCurvatureFR**, we repeat the pairing process with the s_i 's that remain available at this point and without enforcing the constraint that the points of the pair must lie in close proximity of each other. This computation corresponds to step 5. That is, this step pairs each available s_i with the closest available point among the t_j 's even if it is far away from s_i (and is likely to be an unrelated point). This is to ensure that enough distances are returned for their use in the parent protocol. In this step, distances of all unavailable points are set to infinity and each s_i which is still marked as available is updated with the closest distance and the corresponding t_j .

What remains is to discuss protocol **OptimalMotion** that corresponds to secure evaluation of the computation in Algorithm 4 and is given next. The computation in **OptimalMotion** follows the steps of Algorithm 4 and omit its detailed description here. We re-arrange some operations in this protocol to reduce the number of expensive operations.

$$(R, v) \leftarrow \text{OptimalMotion}(\{(t_i = ([x_i], [y_i], [z_i]), s_i = ([x'_i], [y'_i], [z'_i]))\}_{i=1}^n)$$

1. For $i = 1, \dots, n$ do in parallel
 - (a) Compute $[k'_i] = [x_i] \cdot [x'_i] + [y_i] \cdot [y'_i] + [z_i] \cdot [z'_i]$, $[k''_i] = \text{Sqrt}((([x_i]^2 + [y_i]^2 + [z_i]^2)([x'_i]^2 + [y'_i]^2 + [z'_i]^2)))$, and $[k_i] = \text{Div}([k'_i], [k''_i])$.

- (b) Compute $[p_{(i,1)}] = \text{Sqrt}(\frac{1}{2} + \frac{1}{2} \cdot [k_i])$, $[p_{(i,2)}] = \text{Sqrt}(\frac{1}{2} - \frac{1}{2} \cdot [k_i])$.
- (c) Compute $[b_i] = [y_i] \cdot [z'_i] - [z_i] \cdot [y'_i]$, $[b'_i] = [z_i] \cdot [x'_i] - [x_i] \cdot [z'_i]$, and $[b''_i] = [x_i] \cdot [y'_i] - [y_i] \cdot [x'_i]$.
- (d) Compute $[u'_i] = \text{Div}(1, \text{Sqrt}([b_i]^2 + [b'_i]^2 + [b''_i]^2))$ and $u_i = ([u_{(i,1)}], [u_{(i,2)}], [u_{(i,3)}]) = ([b_i] \cdot [u'_i]), ([b'_i] \cdot [u'_i]), ([b''_i] \cdot [u'_i])$.
- (e) Compute and set $q'_i = ([q'_{(i,1)}], [q'_{(i,2)}], [q'_{(i,3)}], [q'_{(i,4)}]) = ([p_{(i,1)}], [p_{(i,2)}] \cdot [u_{(i,1)}], [p_{(i,2)}] \cdot [u_{(i,2)}], [p_{(i,2)}] \cdot [u_{(i,3)}])$.
2. Set $q = ([q_1], [q_2], [q_3], [q_4]) = q'_1$.
3. For $i = 2, \dots, n$ compute $[q''_1] = [q_1] \cdot [q'_{(i,1)}] - [q_2] \cdot [q'_{(i,2)}] - [q_3] \cdot [q'_{(i,3)}] - [q_4] \cdot [q'_{(i,4)}]$, $[q''_2] = [q_1] \cdot [q'_{(i,2)}] + [q'_{(i,1)}] \cdot [q_2] + [q_3] \cdot [q'_{(i,4)}] - [q_4] \cdot [q'_{(i,3)}]$, $[q''_3] = [q_1] \cdot [q'_{(i,3)}] + [q'_{(i,1)}] \cdot [q_3] + [q_2] \cdot [q'_{(i,4)}] - [q_4] \cdot [q'_{(i,2)}]$, and $[q''_4] = [q_1] \cdot [q'_{(i,4)}] + [q'_{(i,1)}] \cdot [q_4] + [q_2] \cdot [q'_{(i,3)}] - [q_3] \cdot [q'_{(i,2)}]$, then set $[q_1] = [q''_1]$, $[q_2] = [q''_2]$, $[q_3] = [q''_3]$, and $[q_4] = [q''_4]$.
4. Compute $[\hat{q}_{(1,1)}] = [q_1]^2$, $[\hat{q}_{(2,2)}] = [q_2]^2$, $[\hat{q}_{(3,3)}] = [q_3]^2$, $[\hat{q}_{(4,4)}] = [q_4]^2$, $[\hat{q}_{(2,3)}] = [q_2] \cdot [q_3]$, $[\hat{q}_{(1,4)}] = [q_1] \cdot [q_4]$, $[\hat{q}_{(2,4)}] = [q_2] \cdot [q_4]$, $[\hat{q}_{(1,3)}] = [q_1] \cdot [q_3]$, $[\hat{q}_{(3,4)}] = [q_3] \cdot [q_4]$, and $[\hat{q}_{(1,2)}] = [q_1] \cdot [q_2]$.
5. Compute matrix $R = \{[r_{ij}]\}_{i,j=1}^3$, where $[r_{11}] = [\hat{q}_{(1,1)}] + [\hat{q}_{(2,2)}] - [\hat{q}_{(3,3)}] - [\hat{q}_{(4,4)}]$, $[r_{12}] = 2 \cdot ([\hat{q}_{(2,3)}] - [\hat{q}_{(1,4)}])$, $[r_{13}] = 2 \cdot ([\hat{q}_{(2,4)}] + [\hat{q}_{(1,3)}])$, $[r_{21}] = 2 \cdot ([\hat{q}_{(2,3)}] + [\hat{q}_{(1,4)}])$, $[r_{22}] = [\hat{q}_{(1,1)}] - [\hat{q}_{(2,2)}] + [\hat{q}_{(3,3)}] - [\hat{q}_{(4,4)}]$, $[r_{23}] = 2 \cdot ([\hat{q}_{(3,4)}] - [\hat{q}_{(1,2)}])$, $[r_{31}] = 2 \cdot ([\hat{q}_{(2,4)}] - [\hat{q}_{(1,3)}])$, $[r_{32}] = 2 \cdot ([\hat{q}_{(3,4)}] - [\hat{q}_{(1,2)}])$, and $[r_{33}] = [\hat{q}_{(1,1)}] - [\hat{q}_{(2,2)}] - [\hat{q}_{(3,3)}] + [\hat{q}_{(4,4)}]$.
6. Compute $[t'_1] = \frac{1}{n} \cdot \sum_{i=1}^n [x_i]$, $[t'_2] = \frac{1}{n} \cdot \sum_{i=1}^n [y_i]$, $[t'_3] = \frac{1}{n} \cdot \sum_{i=1}^n [z_i]$;
7. Compute $[s''_1] = \frac{1}{n} \cdot \sum_{i=1}^n [x'_i]$, $[s''_2] = \frac{1}{n} \cdot \sum_{i=1}^n [y'_i]$, $[s''_3] = \frac{1}{n} \cdot \sum_{i=1}^n [z'_i]$;
8. Compute vector $v = \{[v_i]\}_{i=1}^3$, where $[v_1] = [t'_1] + [r_{11}] \cdot [s''_1] + [r_{12}] \cdot [s''_2] + [r_{13}] \cdot [s''_3]$, $[v_2] = [t'_2] + [r_{21}] \cdot [s''_1] + [r_{22}] \cdot [s''_2] + [r_{23}] \cdot [s''_3]$, $[v_3] = [t'_3] + [r_{31}] \cdot [s''_1] + [r_{32}] \cdot [s''_2] + [r_{33}] \cdot [s''_3]$.
9. Return (R, v) .

Note that here many steps are independent of each other and can be carried out in parallel. For examples steps 1(a)–(b) and 1(c)–(d) correspond to independent branches of computation. Furthermore, some (rather cheap) redundant operations are retained in the protocol for readability, while an implementation would execute them only once. Additional small optimizations are also possible. For example, a number of multiplications in step 1 correspond to multiplication of integer and fixed-point operands, which can be implemented faster than regular fixed-point multiplication.

6.3 Secure Fingerprint Recognition based on a Spectral Minutia Representation

In this section, we present our third secure fingerprint recognition protocol based on spectral minutia representation called **SpectralFR**. The construction builds on Algorithm 5 and incorporates both types of feature reduction (CPCA and LDFT) not included in Algorithm 5. Recall that in the second type of feature reduction, LDFT, (or when both types of feature reduction are applied) rotation of fingerprint/matrix S is performed by multiplying each cell of S by value $e^{-i \frac{2\pi}{N} j \alpha}$, where j is the cell's row and α is the amount of rotation. While it is possible to implement rotations by providing $2\lambda + 1$ different copies of S rotated by different amounts as input into the protocol where λ is the maximum amount of rotation, we perform any necessary rotation inside the protocol to avoid the price of significantly increasing the input size. We were also able to maintain performing only $O(\log \lambda)$ score computations instead of all $2\lambda + 1$ scores in our secure and oblivious protocol, which is an important contribution of this work. Combined with avoiding to increase the input size to have a linear dependency on λ , this allows us to achieve low asymptotic complexity and high efficiency. Low asymptotic complexity is important because the size of fingerprint representation is already large in this approach.

In what follows, we treat the case when $\lambda = 15$ (with the total of 31 rotations to be considered) as in the original algorithm [49], but it is not difficult to generalize the computation to any λ . We apply our generalization and optimization of Algorithm 5 described in section 3.3 with $4 \cdot 3^2 = 36$ different rotations to cover at least 31 necessary alignments. Note that this approach

computes 8 similarity scores instead of 9 in the original algorithm. We number all 36 rotations as $\alpha = -17, \dots, 18$. The rotation constants $e^{-i\frac{2\pi}{N}j\alpha} = \cos(-\frac{2\pi}{N}j\alpha) + i\sin(-\frac{2\pi}{N}j\alpha)$ are fixed and can be precomputed for each $j = 1, \dots, N'$ and $\alpha = -17, \dots, 18$ by each party. We denote these values by public matrix $Z = \{z_{\alpha,j}\}_{\alpha=-17,j=1}^{18,N'}$ which each party stores locally. Each $z_{\alpha,j}$ is a tuple $(z_{\alpha,j}^{(1)} = \cos(-\frac{2\pi}{N}j\alpha), z_{\alpha,j}^{(2)} = \sin(-\frac{2\pi}{N}j\alpha))$, and Z is specified as part of the input in **SpectralFR**.

To be able to compute only $O(\log \lambda)$ scores in the protocol, we need to obviously determine the correct amount of rotation in steps 3 and 4 of Algorithm 5 without revealing any information about k or k' in those steps. We securely realize this functionality by placing the values associated with each possible k or k' in an array and retrieving the right elements at a private index. In more detail, the protocol first computes four scores that correspond to rotations by $-13, -4, 3$, and 12 positions and computes the best among them (steps 3 and 4 below). Because the location of the best score cannot be revealed, in the next step of the algorithm we put together an array consisting of four vectors and one of them is privately selected using **TLookup** (steps 4–5 of the protocol). The selected vector consists of $2N'$ values that allow us to compute two new scores and the maximum score for the next iteration of algorithm. We repeat the process of private retrieval of the necessary rotation coefficients, this time using an array consisting of twelve vectors. After computing two more scores and determining the best score, the algorithm outputs the best score together with the amount of rotation that resulted in that score.

The protocol **SpectralFR** is given next. Because **SpectralFR** corresponds to the computation with both types of feature reduction, input matrices T and S are composed of complex values. Thus, we represent each cell of T as a pair $(a_{i,j}, b_{i,j})$ with the real and imaginary parts, respectively, and each cell of S as a pair $(a'_{i,j}, b'_{i,j})$. All computations proceed over fixed-point values.

$([C_{max}], [\alpha_{max}]) \leftarrow \text{SpectralFR}(T = \{[t_{i,j}] = ([a_{i,j}], [b_{i,j}])\}_{i=1,j=1}^{M',N'}, S = \{[s_{i,j}] = ([a'_{i,j}], [b'_{i,j}])\}_{i=1,j=1}^{M',N'}, Z = \{z_{i,j}\}_{i=-17,j=1}^{18,N'}, \lambda = 15)$

1. Set $[C_{max}] = [0]$.
 2. For $i = 1, \dots, M'$ and $j = 1, \dots, N'$ do in parallel if $(j \neq 1)$ then $[x_{i,j}] = 2([a_{i,j}] \cdot [a'_{i,j}] + [b_{i,j}] \cdot [b'_{i,j}])$, $[y_{i,j}] = 2([a'_{i,j}] \cdot [b_{i,j}] - [a_{i,j}] \cdot [b'_{i,j}])$, else $[x_{i,j}] = [a_{i,j}] \cdot [a'_{i,j}]$, $[y_{i,j}] = -[a_{i,j}] \cdot [b'_{i,j}]$.
 3. For $k = 0, \dots, 3$, do in parallel
 - (a) $[C_{-13+9k}] = 0$.
 - (b) For $i = 1, \dots, M'$ and $j = 1, \dots, N'$ do $[C_{-13+9k}] = [C_{-13+9k}] + z_{-13+9k,j}^{(1)} \cdot [x_{i,j}] + z_{-13+9k,j}^{(2)} \cdot [y_{i,j}]$.
 4. Compute the maximum as $([C_{max}], [\alpha_{max}]) \leftarrow \text{Max}([C_{-13}], [C_{-4}], [C_5], [C_{14}])$.
 5. For $i = 0, \dots, 3$ and $j = 1, \dots, N'$ do in parallel $z'_{i,j} = z_{-16+9i,j}$ and $z'_{i,N'+j} = z_{-10+9i,j}$.
 6. Let $Z'_i = (z'_{i,1}, \dots, z'_{i,2N'})$ for $i = 0, \dots, 3$ and execute $[Z'_{max}] \leftarrow \text{TLookup}((Z'_0, \dots, Z'_3), [\alpha_{max}])$; let $[Z'_{max}] = ([\hat{z}_1], \dots, [\hat{z}_{2N'}])$.
 7. For $i = 1, \dots, M'$ and $j = 1, \dots, N'$ do $[C_{\alpha_{max}-3}] = [C_{\alpha_{max}-3}] + [\hat{z}_j^{(1)}] \cdot [x_{i,j}] + [\hat{z}_j^{(2)}] \cdot [y_{i,j}]$, $[C_{\alpha_{max}+3}] = [C_{\alpha_{max}+3}] + [\hat{z}_{N'+j}^{(1)}] \cdot [x_{i,j}] + [\hat{z}_{N'+j}^{(2)}] \cdot [y_{i,j}]$.
 8. Compute $([C_{max}], [\alpha_{max}]) \leftarrow \text{Max}([C_{\alpha_{max}-3}], [C_{max}], [C_{\alpha_{max}+3}])$.
 9. For $i = 0, \dots, 11$ and $j = 1, \dots, N'$ do in parallel $z'_{i,j} = z_{-17+3i,j}$ and $z'_{i,N'+j} = z_{-15+3i,j}$.
 10. Let $Z'_i = (z'_{i,1}, \dots, z'_{i,2N'})$ for $i = 0, \dots, 11$ and execute $[Z'_{max}] \leftarrow \text{TLookup}((Z'_0, \dots, Z'_{11}), [\alpha_{max}])$; let $[Z'_{max}] = ([\hat{z}_1], \dots, [\hat{z}_{2N'}])$.
 11. For $i = 1, \dots, M'$ and $j = 1, \dots, N'$ do $[C_{\alpha_{max}-1}] = [C_{\alpha_{max}-1}] + [\hat{z}_j^{(1)}] \cdot [x_{i,j}] + [\hat{z}_j^{(2)}] \cdot [y_{i,j}]$, $[C_{\alpha_{max}+1}] = [C_{\alpha_{max}+1}] + [\hat{z}_{N'+j}^{(1)}] \cdot [x_{i,j}] + [\hat{z}_{N'+j}^{(2)}] \cdot [y_{i,j}]$.
 12. Compute $([C_{max}], [\alpha_{max}]) \leftarrow \text{Max}([C_{\alpha_{max}-1}], [C_{max}], [C_{\alpha_{max}+1}])$.
 13. Return $[C_{max}]$ and $[\alpha_{max}]$.
-

For efficiency reasons, we perform all multiplications associated with the score computation without any rotation in the beginning of the protocol (step 1). This computation (i.e., multiplications of

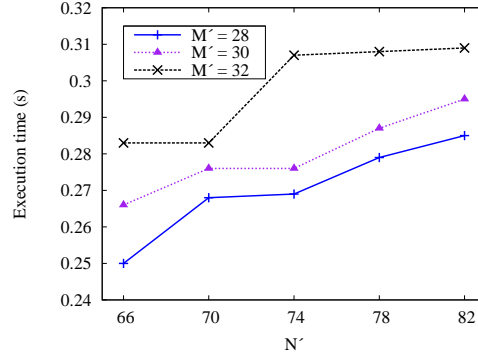


Figure 2: Execution time of **SpectralFR** in seconds in the secret sharing multi-party setting.

the real and imaginary components of each $t_{i,j}$ and $s_{i,j}$) is common to all score computations (see equation 1) and is reused later in the protocol. Then to compute a score between T and S rotated by α positions, the coefficients from Z are multiplied to the computed products. In particular, the computation takes form of $\{[z_{\alpha,j}^{(1)}] \cdot [x_{i,j}]\}_{i=1,j=1}^{M',N'}$ and $\{[z_{\alpha,j}^{(2)}] \cdot [y_{i,j}]\}_{i=1,j=1}^{M',N'}$ according to equation 1, which are added together to get the score. The rest of the protocol proceeds as described above by using private table lookups twice. Note that the result of each table lookup is an array as opposed to a single element. Also note that the score computation uses public $z_{i,j}$'s in step 1(b), but the coefficients become private in steps 7 and 11 because they depend of private data. Finally, the factor $\frac{1}{MN^2}$ present in equation 1 is not included in the computation because it is public. The computed score can be scaled down by this factor by an output recipient if necessary.

Recall that **SpectralFR** uses fixed-point arithmetic, but performance of some operations can be optimized. For example, in the SS setting, we can skip truncation after each multiplication in step 7 or 11 and instead truncate the sum after adding all products. In addition, we can restrict variables to shorter bitlength representations when the range of values they store is known to be small. This is relevant to multiplications in step 2 in the GC setting, where $t_{i,j}$ and $s_{i,j}$ are known not to exceed $N + 1$ and can be represented using a short bitlength for the integer part.

7 Performance Evaluation

In this section we evaluate performance of the spectral minutia representation fingerprint recognition protocol **SpectralFR** to demonstrate that the proposed protocols are practical. We provide experimental results for both two-party and multi-party settings. The implementations were written in C/C++. In the two-party setting, our implementation uses the JustGarble library [10, 9] for circuit garbling and GC evaluation which we modified to support the half-gates optimization [52]. In the multi-party settings, we use PICCO [53] with three computational parties, which utilized linear threshold SS.

In our implementation, we assume that the inputs T and S are represented with 32-bit precision after the radix point. Our GC implementation maintains 56-bit fixed-point representation (24 and 32 bits before and after the radix point), while in the SS setting we let the numbers grow beyond the 56 bits to avoid the cost of truncation, but perform truncation prior to returning the result. All machines used in the experiments had identical hardware with four-core 3.2GHz Intel i5-3470

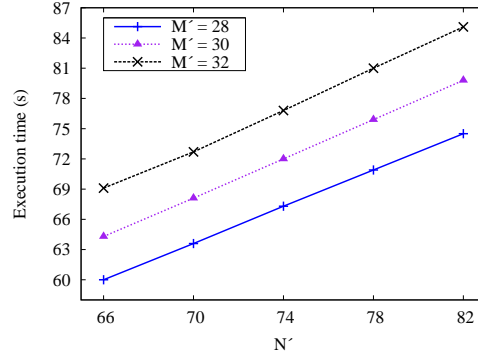


Figure 3: Execution time of **SpectralFR** in seconds in the garbled circuits two-party setting.

M'	N'				
	66	70	74	78	82
28	0.250	0.268	0.269	0.279	0.285
30	0.266	0.276	0.276	0.287	0.295
32	0.283	0.283	0.307	0.308	0.309

Table 3: Execution time of protocol **SpectralFR** in seconds using secret sharing.

processors running Red Hat Linux 2.6.32 and were connected via a 1Gb LAN. Only one core was used during the experiments. We ran each experiment 10 times and report the mean value.

The results of **SpectralFR** performance evaluation can be found in Table 3 and Figure 2 for the three-party case and in Table 4 and Figure 3 for the two-party case. We report performance for a range of parameters N' and M' that might be used in practice. All numbers correspond to the overall runtime including communication. Because in the GC setting the overall runtime is composed of multiple components, we provide a breakdown of the total time according to its constituents. We have that garbling uses 26–27% of the overall time, GC evaluation takes 15–16%, oblivious transfer takes about 1% of the time, and communication time is about 57%. The portion of non-XOR gates in the circuits was about 27.6%. These numbers tell us that communication takes most of the time (even with the half-gates optimization that reduces communication). Because circuit garbling can be performed offline, the work associated with circuit garbling (a quarter of the overall time) and GC transmission can be done in advance saving most of the total time.

As one can see, secure fingerprint recognition takes a fraction of a second in the multi-party setting and tens of seconds in the two-party setting. This shows that the protocol is efficient for practical use. The computation used in **SpectralFR** is a rare example of a functionality that can be implemented significantly more efficiently using SS techniques than GC evaluation due to its heavy use of multiplications.

8 Conclusions

In this work, we design three secure and efficient protocols for fingerprint alignment and matching for two-party and multi-party settings. They are based on popular algorithms in the computer vision literature and employ new non-trivial techniques. The constructions are presented in the

M'	N'									
	66		70		74		78		82	
	Time	Gates	Time	Gates	Time	Gates	Time	Gates	Time	Gates
28	60.0	454.2M	63.6	481.8M	67.3	509.4M	70.9	537.0M	74.5	564.5M
30	64.3	486.7M	68.1	516.2M	72.0	545.8M	75.9	575.3	79.8	604.8M
32	69.1	519.1M	72.7	550.6M	76.8	582.1M	81.0	613.6	85.1	645.1M

Table 4: Execution time of protocol **SpectralFR** in seconds and the total number of gates in millions in its implementation using GCs.

semi-honest setting and known results can be applied to strengthen the security to sustain malicious actors. We believe that this is the first work that treats the problem of secure fingerprint alignment.

Another main contribution of this work is designing secure constructions for fundamental numeric and set operations. We present novel secure protocols for sine, cosine, arctangent, and square root for fixed-point numbers, as well as a novel secure and data-oblivious protocol for selection of the f th smallest element of a set (for any type of data). The techniques are applicable to both two-party and multi-party settings. Our hope is that these techniques find their applicability well beyond fingerprint recognition and that this work will facilitate secure processing of biometric data in practice.

Acknowledgments

The authors would like to thank Karthik Nandakumar for his help with making the source code of the fingerprint recognition algorithm from [44] available. This work was supported in part by grants 1223699, 1228639, 1319090, and 1526631 from the National Science Foundation and FA9550-13-1-0066 from the Air Force Office of Scientific Research, as well as DARPA agreement no. AFRL FA8750-15-2-0092. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies or the U.S. Department of Defense.

References

- [1] Fingerprint minutiae viewer (FpMV). <https://www.nist.gov/services-resources/software/fingerprint-minutiae-viewer-fpmv>.
- [2] NIST special database 4. <https://www.nist.gov/srd/nist-special-database-4>.
- [3] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele. Secure computation on floating point numbers. In *Network and Distributed Security Symposium (NDSS)*, 2013.
- [4] G. Asharov, Y. Lindell, and T. Rabin. Perfectly-secure multiplication for any $t < n/3$. In *Advances in Cryptology – CRYPTO*, pages 240–258, 2011.
- [5] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 535–548, 2013.
- [6] M. Atallah, M. Bykova, J. Li, K. Friksen, and M. Topkara. Private collaborative forecasting and benchmarking. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 103–114, 2004.

- [7] M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, F. Scotti, and A. Piva. Privacy-preserving FingerCode authentication. In *ACM Workshop on Multimedia and Security (MM&Sec)*, pages 231–240, 2010.
- [8] K. Batchier. Sorting networks and their applications. In *AFIPS Spring Joint Computer Conference*, pages 307–314, 1968.
- [9] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. The JustGarble library. <http://cseweb.ucsd.edu/groups/justgarble/>.
- [10] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium of Security and Privacy*, pages 478–492, 2013.
- [11] P. Besl and N. McKay. Method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.
- [12] M. Blanton. Empirical evaluation of secure two-party computation models. Technical Report TR 2005-58, CERIAS, Purdue University, 2005.
- [13] M. Blanton. Achieving full security in privacy-preserving data mining. In *IEEE International Conference on Information Privacy, Security, Risk and Trust*, pages 925–934, 2011.
- [14] M. Blanton and E. Aguiar. Private and oblivious set and multiset operations. *International Journal of Information Security*, 15(5):493–518, Oct. 2016.
- [15] M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *European Symposium on Research in Computer Security (ESORICS)*, pages 190–209, 2011.
- [16] M. Blanton and P. Gasti. Secure and efficient iris and fingerprint identification. In D. Ngo, A. Teoh, and J. Hu, editors, *Biometric Security*, chapter 9. Cambridge Scholars Publishing, 2015.
- [17] M. Blanton and S. Saraph. Oblivious maximum bipartite matching size algorithm with applications to secure fingerprint identification. In *European Symposium on Research in Computer Security (ESORICS)*, pages 384–406, 2015.
- [18] D. Bogdanov, S. Laur, and J. Willemsen. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, pages 192–206, 2008.
- [19] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
- [20] O. Catrina and S. D. Hoogh. Improved primitives for secure multiparty integer computation. In *Security and Cryptography for Networks*, pages 182–199. Springer, 2010.
- [21] O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *In Financial Cryptography and Data Security*, pages 35–50, 2010.
- [22] D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek. The trimmed iterative closest point algorithm. In *International Conference on Pattern Recognition*, pages 545–548, 2002.
- [23] I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Advances in Cryptology – EUROCRYPT*, pages 445–465, 2010.

- [24] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.
- [25] W. Du and M. J. Atallah. Secure multi-party computation problems and their applications: A review and open problems. In *ACM Workshop on New Security Paradigms (NSPW)*, pages 13–22, 2001.
- [26] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 235–253, 2009.
- [27] R. Gamboa and J. Cowles. Formal verification of medina’s sequence of polynomials for approximating arctangent. *arXiv preprint arXiv:1406.1561*, 2014.
- [28] R. Gennaro, M. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 101–111, 1998.
- [29] M. T. Goodrich. Data-oblivious external-memory algorithms for the compaction, selection, and sorting of outsourced data. In *ACM Symposium on Parallelism in Algorithms and Architectures*, pages 379–388, 2011.
- [30] J. Hart, E. Cheney, C. Lawson, H. Maehly, C. Mesztenyi, J. Rice, H. Thacher, and C. Witzgall. *Computer approximations*. John Wiley & Sons, Inc., 1968.
- [31] W. Henecka, K. S. A. R. Sadeghi, T. Schneider, and I. Wehrenberg. Tasty: Tool for automating secure two-party computations. In *ACM Conference on Computer and Communications Security*, pages 451–462. ACM, 2010.
- [32] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.
- [33] Y. Huang, L. Malka, D. Evans, and J. Katz. Efficient privacy-preserving biometric identification. In *Network and Distributed System Security Symposium (NDSS)*, 2011.
- [34] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, pages 145–161, 2003.
- [35] A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. *Doklady Akademii Nauk SSSR*, 145:293–294, 1962.
- [36] F. Kerschbaum, M. Atallah, D. M’Raïhi, and J. Rice. Private fingerprint verification without local storage. In *International Conference on Biometric Authentication (ICBA)*, pages 387–394, 2004.
- [37] V. Kolesnikov, A. R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Cryptology and Network Security*, pages 1–20. Springer, 2009.
- [38] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 486–498, 2008.

- [39] B. Kreuter, a. shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security Symposium*, 2012.
- [40] M. Liedel. Secure distributed computation of the square root and applications. In *Information Security Practice and Experience*, pages 277–288. Springer, 2012.
- [41] Y. Lindell and B. Pinkas. A proof of security of Yao’s protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.
- [42] D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar. *Handbook of Fingerprint Recognition*. New York Springer-Verlag, 2003.
- [43] H. A. Medina. A sequence of polynomials for approximating arctangent. *The American Mathematical Monthly*, 113(2):156–161, 2006.
- [44] K. Nandakumar, A. K. Jain, and S. Pankanti. Fingerprint-based fuzzy vault: Implementation and performance. *IEEE Transactions on Information Forensics and Security*, 2(4):744–757, 2007.
- [45] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 448–457, 2001.
- [46] A. Naveh and E. Tromer. PhotoProof: Cryptographic image authentication for any set of permissible transformations. In *IEEE Symposium on Security and Privacy*, pages 255–271, 2016.
- [47] S. Shahandashti, R. Safavi-Naini, and P. Ogunbona. Private fingerprint matching. In *Australasian Conference on Information Security and Privacy*, pages 426–433, 2012.
- [48] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
- [49] H. Xu, R. Veldhuis, A. Bazen, T. Kevenaar, T. Akkermans, and B. Gokberk. Fingerprint verification using spectral minutiae representations. *IEEE Transactions on Information Forensics and Security*, 4(3):397–409, 2009.
- [50] H. Xu, R. Veldhuis, T. Kevenaar, and T. Akkermans. A fast minutiae-based fingerprint recognition system. *IEEE Systems Journal*, 3(4):418–427, 2009.
- [51] A. Yao. How to generate and exchange secrets. In *IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.
- [52] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, pages 220–250, 2015.
- [53] Y. Zhang, A. Steele, and M. Blanton. Picco: A general-purpose compiler for private distributed computation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 813–826, 2013.

A Alternative Design of Building Blocks

A.1 Trigonometric Functions

Recall that in section 5.2.1 we provide a method for securely evaluating trigonometric and inverse trigonometric functions using polynomial evaluation. In this section we present an alternative approach based on private table lookup. When the input range is constrained (as in the fingerprint recognition algorithms we are considering), this mechanism could result in improved performance, but in general applications with arbitrary precision it is likely to run into scalability issues.

Consider a trigonometric function (such as sine or cosine used in this work) that needs to be evaluated on private input a . Let the value of a be in the range $[a_{min}, a_{max}]$ with N denoting the number of the elements in the range. Then the approach consists of precomputing the function on all possible inputs and storing the result in an array $Z = (z_0, \dots, z_{N-1})$. Consequently, evaluation of the function on private a corresponds to privately retrieving the needed element of the array Z using a to determine the index. This procedure is formalized in the protocol **Trig** below.

$[b] \leftarrow \text{Trig}([a], Z = \{z_i\}_{i=0}^{N-1}, a_{min})$

1. Compute $[b] \leftarrow \text{TLookup}(\langle z_0, \dots, z_{N-1} \rangle, [a] - a_{min})$.
 2. Return $[b]$.
-

Complexity of this protocol is heavily dominated by that of **TLookup** on an array of size N , and the overall complexity is thus $O(N \log N)$. In our application, when we compute image rotation on a small angle expressed as an integer in degrees, the value of N is small. For example, with rotations by at most 10 degrees in each direction, we have $N = 21$ and thus this approach becomes very efficient.

The situation, however, is different with inverse trigonometric functions. The input to such functions is now a real number, which changes the way a table lookup is to be implemented; but more importantly, it is desirable to have the function support finer granularity than outputting integer angles expressed in degrees. This will require the size of the array used for private table lookup increase substantially, mitigating the benefits of this approach. Thus, for inverse trigonometric functions such as arctangent we recommend our main solution described in section 5.2.1.

A.2 Square Root

As an alternative approach to securely computing the square root of $[a]$, we suggest to use approximations from [30, Chapter 6]. [30] uses polynomial approximations for square root based on Heron's and Newton's iterations and has polynomial coefficients precomputed for different input ranges and precision values.

To work correctly, this method requires that the range of input $[a]$ is reduced to a specific range. Range reduction of a follows the formula:

$$\sqrt{x} = \beta^v \sqrt{a\beta^{-2v}} = \beta^v \sqrt{\beta} \sqrt{a\beta^{-(2v+1)}} \quad (2)$$

Based on [30, Chapter 6], the constant β can be set to the base of the computer number system (such as 2) and the range of the input will be reduced to $\frac{1}{\beta} \leq a' \leq 1$, where $a' = a\beta^{-2v}$ or $a' = a\beta^{-(2v+1)}$ is the normalized input. Alternatively, if a longer range can give acceptable accuracy, the constant β can be set to a power of the computer number system base (e.g., 2^2 , etc.) and the range will still be reduced to $\frac{1}{\beta} \leq a' \leq 1$. The trade-off here is that the larger the value of β is, the smaller precision can be obtained using polynomials of the same degree. Thus, to achieve equivalent precision,

polynomials of larger degrees will need to be used with larger β compared to lower values of β . We suggest to set $\beta = 2$ and thus reduce the range of the input to $\frac{1}{2} \leq a' \leq 1$, which provides high precision for given polynomial degrees.

Next, note that if we let $\frac{1}{2} \leq a' < 1$, the normalized value a' lies in the same range as the normalized value in Section 5.2.2 and we can apply the normalization procedure **Norm** described in that section. Furthermore, if we let $\beta = 2$ and adopt notation $a' = a \cdot 2^{-w}$ used in Section 5.2.2, then based on equation 2, 2^{-w} is either 2^{-2v} or $2^{-(2v+1)}$ depending on whether w is even or odd. Recall that the **Norm** protocol outputs $[a']$ together with $[2^{\lfloor \frac{w}{2} \rfloor}]$ and bit $[c]$ which is set when w is odd. This is very similar to what want here. That is, $2^v = 2^{\lfloor \frac{w}{2} \rfloor}$ and c is set when w is odd to indicate that multiplication by $\sqrt{2}$ of the result is needed (as in equation 2). This means that we can utilize a variant of the **Norm** protocol from Section 5.2.2 that outputs $[2^{\lfloor \frac{w}{2} \rfloor}]$ instead of $[2^{-\lfloor \frac{w}{2} \rfloor}]$ but otherwise operates in the same way. To make this change to the **Norm** protocol of Section 5.2.2, we only need to replace step 9 with $[2^{\lfloor \frac{w}{2} \rfloor}] \leftarrow (d_{\ell-1}, \dots, [d_0]) = (0^{\frac{\ell-k}{2}-1}, [u_{\frac{\ell}{2}}], [u_{\frac{\ell}{2}-1}], \dots, [u_0], 0^{\frac{k}{2}})$ and output $[2^{\lfloor \frac{w}{2} \rfloor}]$ in step 11. It is also not difficult to modify the normalization protocol in [40] for the SS setting. In particular, we need to replace lines 8–10 of **NormSQ** in [40] by the following computations:

8. Let $[u_0] = [z_0]$, $[u_{\frac{k}{2}}] = [z_{k-1}]$ and for $i = 1, \dots, \frac{k}{2} - 1$ do in parallel $[u_i] = [z_{2i-1}] + [z_{2i}]$.
9. Compute $[2^{\lfloor \frac{w}{2} \rfloor}] = \sum_{i=0}^{k/2} 2^{i+f/2} [u_i]$.
10. Set $[d] = \sum_{i=0}^{\frac{k}{2}-1} [z_{2i}]$.

The protocol then returns $\langle [c], [2^{\lfloor \frac{w}{2} \rfloor}], [d] \rangle$, which corresponds to $\langle [a'], [2^{\lfloor \frac{w}{2} \rfloor}], [c] \rangle$ using our notation from **Norm**. Note that here a fixed-point value is represented using the total of k bits, f of which are stored after the radix point.

Once the input is normalized to the desired range, evaluation of the square root function amounts to evaluating polynomials on the normalized input. Given the desired precision of the computation in bits, we determine the minimum degrees of the polynomials and the coefficients corresponding to the polynomials can be looked up from [30]. Similar to the polynomial evaluation approach described for trigonometric functions in Section 5.2.1, there are two types of polynomial approximations: One type uses a regular polynomial $P(x)$ of degree N and the other type uses a rational function $P(x)/Q(x)$, where P and Q are polynomials of degrees N and M , respectively. Based on the cost of multiplication and division in the selected setting, either a regular polynomial or rational function might be preferred. We recommend to use a rational function for GCs-based implementations and regular polynomial for SS-based implementations. The overall square root protocol **Sqrt2** is given next.

$[b] \leftarrow \text{Sqrt2}([a])$

1. Execute $\langle [a'], [2^{\lfloor \frac{w}{2} \rfloor}], [c] \rangle \leftarrow \text{Norm}([a])$.
 2. Lookup the minimum polynomial degrees N and M using range $[\frac{1}{2}, 1)$ for which precision of the approximation is at least k bits.
 3. Lookup polynomial coefficients p_0, \dots, p_N and q_0, \dots, q_M for square root approximation.
 4. Compute $([z_1], \dots, [z_{\max(N,M)}]) \leftarrow \text{PreMul}([a'], \max(N, M))$.
 5. Compute $[y_P] = p_0 + \sum_{i=1}^N p_i [z_i]$.
 6. Compute $[y_Q] = q_0 + \sum_{i=1}^M q_i [z_i]$.
 7. Compute $[y] \leftarrow \text{Div}([y_P], [y_Q])$.
 8. Compute and return $[b] = [y] \cdot [2^{\lfloor \frac{w}{2} \rfloor}] \cdot (([c] \wedge \sqrt{2}) \oplus \neg[c])$.
-

In step 8 of this protocol we use (cheap) XOR in the GC setting, while in the SS setting, that

operation is replaced with addition. As before, this protocol is written using a rational function for approximation. When a single polynomial is used instead, the protocol evaluates only a single polynomial P of (a different) degree N and the division operation is skipped in step 7.